Home FAQ Apps API Protocol Twitter

Telegram

a new era of messaging

Recent News

Sep 5
Scheduled Messages, Reminders, Custom Cloud Themes and More Privacy

Aug 15
Celebrating 6 Years of Telegram

Aug 9
Silent Messages, Slow Mode, Admin Titles and More

Telegram for **Android**

Telegram for **iPhone** / **iPad**

Telegram for **WP**

A native app for every platform

Telegram for **PC/Mac/Linux**

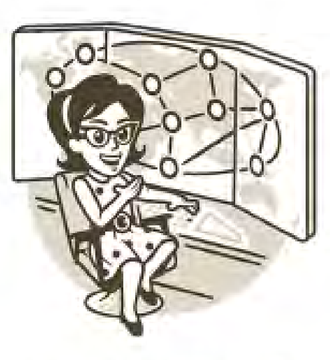
Telegram for **macOS**

Why switch to Telegram?

Private
Telegram messages are heavily encrypted and can self-destruct.

Cloud-Based
Telegram lets you access your messages from multiple devices.

Fast
Telegram delivers messages faster than any other application.


Distributed
Telegram servers are spread worldwide for security and speed.

Open
Telegram has an open API and protocol free for everyone.

Free
Telegram is free forever. No ads. No subscription fees.

Secure
Telegram keeps your messages safe from hacker attacks.

Powerful
Telegram has no limits on the size of your media and chats.

We Can do It!
Help make messaging safe again – spread the word about **Telegram**.

What can you do with Telegram?

Connect
from most remote locations.

Coordinate
groups of up to 200,000 members.

Synchronize
your chats across all your devices.

Send
documents of any type.

Encrypt
personal and business secrets.

Destruct
your messages with a timer.

Store
your media in the cloud.

Build
your own tools on our API.

Enjoy
feedback from your customers.

Telegram
Telegram is a cloud-based mobile and desktop messaging app with a focus on security and speed.

About
FAQ
Blog
Jobs

Mobile Apps
iPhone/iPad
Android
Windows Phone

Desktop Apps
PC/Mac/Linux
macOS
Web-browser

Platform
API
Translations
Instant View

Telegram FAQ

Language

> This FAQ provides answers to basic questions about Telegram.
> Check out our Advanced FAQ for more technical information.

General Questions

Q: What is Telegram? What do I do here?

Telegram is a messaging app with a focus on speed and security, it's super-fast, simple and free. You can use Telegram on all your devices **at the same time** — your messages sync seamlessly across any number of your phones, tablets or computers.

With Telegram, you can send messages, photos, videos and **files** of any type (doc, zip, mp3, etc), as well as create groups for up to **200,000** people or channels for broadcasting to **unlimited** audiences. You can write to your phone contacts and find people by their **usernames**. As a result, Telegram is like SMS and email combined — and can take care of all your personal or business messaging needs. In addition to this, we support end-to-end encrypted voice calls.

Q: Who is Telegram for?

Telegram is for everyone who wants fast and reliable messaging and calls. Business users and small teams may like the large groups, usernames, desktop apps and powerful file sharing options.

Since Telegram groups can have up to 200,000 members, we support replies, mentions and hashtags that help maintain order and keep communication in large communities efficient. You can appoint admins with advanced tools to help these communities prosper in peace. Public groups can be joined by anyone and are powerful platforms for discussions and collecting feedback.

In case you're more into pictures, Telegram has animated gif search, a state of the art photo editor, and an open sticker platform (find some cool stickers here or here). What's more, there is no need to worry about disk space on your device. With Telegram's cloud support and cache management options, Telegram can take up nearly **zero** space on your phone.

Those looking for extra privacy should check out our advanced settings and rather revolutionary policy. And if you want secrecy, try our device-specific Secret Chats with self-destructing messages, photos, and videos — and lock your app with an additional passcode.

> We keep evolving — check out the blog and follow us on twitter and Telegram to stay in touch.

Q: How is Telegram different from WhatsApp?

Unlike WhatsApp, Telegram is a cloud-based messenger with seamless sync. As a result, you can access your messages from several devices at once, including tablets and computers, and share an unlimited number of photos, videos and files (doc, zip, mp3, etc.) of up to 1,5 GB *each*. And if you don't want to store all that data on your device, you can always keep it in the cloud.

Thanks to our multi-data center infrastructure and encryption, Telegram is faster and way more secure. On top of that, Telegram is free and will stay free — no ads, no subscription fees, forever.

Our API is open, and we welcome developers to create their own Telegram apps. We also have a Bot API, a platform for developers that allows anyone to easily build specialized tools for Telegram, integrate any services, and even accept payments from users around the world.

And that's just the tip of the iceberg. Don't forget to check out this paragraph for even more exclusive stuff.

Q: How old is Telegram?

Telegram for iOS was launched on August 14, 2013. The alpha version of Telegram for Android officially launched on October 20, 2013. More and more Telegram clients appear, built by independent developers using Telegram's open platform.

Q: Which devices can I use?

You can use Telegram on smartphones, tablets, and even computers. We have apps for iOS (8.0 and above), Android (4.1 and up) and Windows Phone. You can also use Telegram's web version or install one of our desktop apps for Windows, macOS, and Linux.

> You can log in to Telegram from as many of your devices as you like — all **at the same time**. Just use your main mobile phone number to log in everywhere.

Our API is open for developers, should you want to build your own applications for other platforms.

Q: Who are the people behind Telegram?

Telegram is supported by Pavel Durov and his brother Nikolai. Pavel supports Telegram financially and ideologically while Nikolai's input is technological. To make Telegram possible, Nikolai developed a unique custom data protocol, which is open, secure and optimized for work with multiple data-centers. As a result, Telegram combines security, reliability and speed on any network.

> See also: articles about Telegram

Q: Where is Telegram based?

The Telegram development team is based in Dubai.

Most of the developers behind Telegram originally come from St. Petersburg, the city famous for its unprecedented number of highly skilled engineers. The Telegram team had to leave Russia due to local IT regulations and has tried a number of locations as its base, including Berlin, London and Singapore. We're currently happy with Dubai, although are ready to relocate again if local regulations change.

Q: Will you have ads? Or sell my data? Or steal my beloved and enslave my children?

No.

Q: How are you going to make money out of this?

We believe in fast and secure messaging that is also 100% free.

Pavel Durov, who shares our vision, supplied Telegram with a generous donation, so we have quite enough money for the time being. If Telegram runs out, we will introduce non-essential paid options to support the infrastructure and finance developer salaries. But making profits will never be an end-goal for Telegram.

Q: What are your thoughts on internet privacy?

Big internet companies like Facebook or Google have effectively hijacked the privacy discourse in the recent years. Their marketers managed to convince the public that the most important things about privacy are superficial tools that allow hiding your public posts from the people around you. Adding these superficial tools enables companies to calm down the public and change nothing in how they are turning over private data to marketers and other third parties.

At Telegram we think that the two most important components of Internet privacy should be instead:

1. Protecting your private conversations from snooping third parties, such as officials, employers, etc.
2. Protecting your personal data from third parties, such as marketers, advertisers, etc.

This is what everybody should care about, and these are some of our top priorities. Telegram's aim is to create a truly free messenger, without the usual caveats. This means that instead of diverting public attention with lowimpact settings, we can afford to focus on the real privacy issues that exist in the modern world.

Q: What about GDPR?

New regulations regarding data privacy called the General Data Protection Regulation (GDPR) came into force in Europe on May 25, 2018. Since taking back our right to privacy was the reason we made Telegram, there wasn't much we had to change. We don't use your data for ad targeting, we don't sell it to others, and we're not part of any ~~mafia family~~ "family of companies."

Telegram only keeps the information it needs to function as a feature-rich cloud service — for example, your cloud chats so that you can access them from any devices without using third-party backups, or your contacts so that you can rely on your existing social graph when messaging people on Telegram. Please see our Privacy Policy for more information.

You can use **@GDPRbot** to:

- Request a copy of all your data that Telegram stores.
- Contact us about Data Privacy.

Android users got a GDPR update with version 4.8.9 which allows more control over synced contacts and adds other privacy settings. On June, 1, Apple approved Telegram v.4.8.2 for iOS with these features.

Q: There's illegal content on Telegram. How do I take it down?

All Telegram chats and group chats are private amongst their participants. We do not process any requests related to them.

But **sticker sets**, **channels**, and **bots** on Telegram are *publicly available*. If you find sticker sets or bots on Telegram that you think are illegal, please ping us at abuse@telegram.org.

You can also use the 'report' buttons right inside our apps, see this post on our official @ISISwatch channel for details.

> Note: If a scammer is pretending to be you, contact @NoToScam

Q: A bot or channel is infringing on my copyright. What do I do?

All Telegram chats and group chats are private amongst their participants. We do not process any requests related to them. But **sticker sets**, **channels**, and **bots** on Telegram are *publicly available*.

If you see a bot, channel, or sticker set that is infringing on your copyright, kindly submit a complaint to dmca@telegram.org. Please note that such requests should only be submitted by the copyright owner or an agent authorized to act on the owner's behalf.

Q: Wait! 0_o Do you process take-down requests from third parties?

Our mission is to provide a secure means of communication that works everywhere on the planet. To do this in the places where it is most needed (and to continue distributing Telegram through the App Store and Google Play), we have to process legitimate requests to take down illegal **public** content (e.g., sticker sets, bots, and channels) within the app. For example, we can take down sticker sets that violate intellectual property rights or porn bots.

User-uploaded stickers sets, channels, and bots by third-party developers are not part of the core Telegram UI. Whenever we receive a complaint at abuse@telegram.org or dmca@telegram.org regarding the legality of public content, we perform the necessary legal checks and take it down when deemed appropriate.

Please note that this does **not** apply to local restrictions on freedom of speech. For example, if criticizing the government is illegal in some country, Telegram won't be a part of such politically motivated censorship. This goes against our founders' principles. While we do block terrorist (e.g. ISIS-related) bots and channels, we will not block anybody who peacefully expresses alternative opinions.

Q: My bot or sticker set was banned unfairly, what do I do?

If you think we banned your bot, channel, or sticker set for no apparent reasons, drop us a line at abuse@telegram.org.

Q: Do you process data requests?

Secret chats use end-to-end encryption, thanks to which we don't have any data to disclose.

To protect the data that is not covered by end-to-end encryption, Telegram uses a distributed infrastructure. Cloud chat data is stored in multiple data centers around the globe that are controlled by different legal entities spread across different jurisdictions. The relevant decryption keys are split into parts and are never kept in the same place as the data they protect. As a result, several court orders from different jurisdictions are required to force us to give up any data.

Thanks to this structure, we can ensure that no single government or block of like-minded countries can intrude on people's privacy and freedom of expression. Telegram can be forced to give up data only if an issue is grave and universal enough to pass the scrutiny of several different legal systems around the world.

To this day, we have disclosed 0 bytes of user data to third parties, including governments.

Telegram Basics

Q: Who can I write to?

You can write to people who are in your phone contacts and have Telegram. Another way of contacting people is to type their Telegram username into the search field.

You can set a public username for your Telegram account too. Other people will be able to search and find you by that username — and send messages to you even if they don't know your number. You can learn more about usernames here.

Q: How do I know who in my contacts has Telegram?

Your contacts, who have Telegram, are shown at the top of your Contacts. They also have pictures.

Q: How do I invite my friends?

iOS: The basic invitations are simple SMS messages. They will be charged as standard outgoing SMS by your carrier (unless sent via iMessage). Naturally, you have other options to bring your friends here. Try sending them a download link via any other messaging service: email, Facebook, WhatsApp, an actual telegram — you name it. The link: **https://telegram.org/dl/**

Android: Open the app menu (swipe right in chat list) — Invite Friends. Then choose an application via which you would like to send out invitations.

Windows Phone: Go to Contacts — Invite Friends. Then choose an application via which you would like to send out invitations.

> You can give your friends a t.me link with your username so that they can easily find you on Telegram even if they don't have your phone number.

Q: What do the green checks mean?

One check — message delivered to the Telegram cloud and your friend has been notified if he allows notifications.
Two checks — message read (your friend opened Telegram and opened the conversation with the message).

We don't have a 'delivered to device' status for messages because Telegram can run on as many devices as you want. So which particular one would that check mean?

Q: Can I hide my 'last seen' time?

You can choose who sees this info in Privacy and Security settings.

Remember that you won't see Last Seen timestamps for people with whom you don't share your own. You will, however, see an approximate last seen value. This keeps stalkers away but makes it possible to understand whether a person is reachable over Telegram. There are four possible approximate values:

- **Last seen recently** — covers anything between 1 second and 2-3 days

- **Last seen within a week** — between 2-3 and seven days
- **Last seen within a month** — between 6-7 days and a month
- **Last seen a long time ago** — more than a month (this is also always shown to blocked users)

Q: Who can see me 'online'?

The last seen rules apply to your online status as well. People can only see you online if you're sharing your last seen status with them.

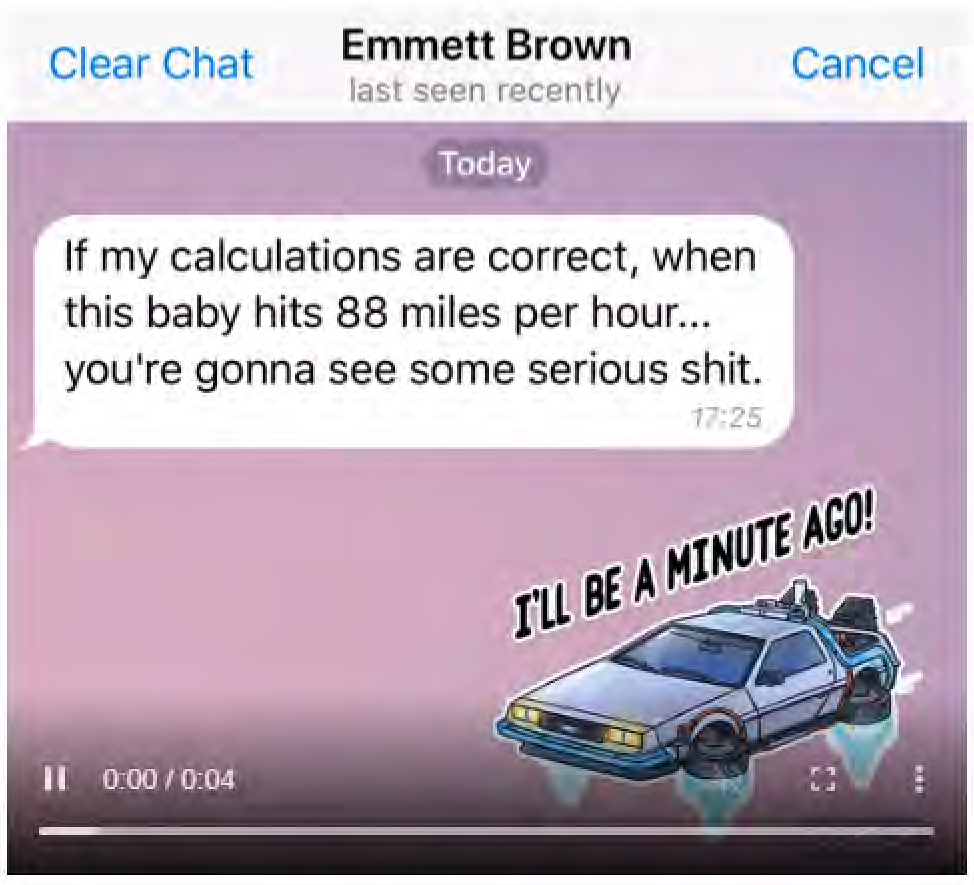
There are some exceptions because sometimes it is obvious that you are online. Regardless of the last seen settings, people will see you online for a brief period (~30 seconds) if you do the following:

- Send them a message in a one-on-one chat or in a group where you both are members.
- Read a message they sent you in a one-on-one chat.
- Broadcast a "typing…" status to their chat with you or to a group where you both are members.

If you're not sharing your last seen timestamp with someone and don't do anything of the above, they'll never see you online. Another way of achieving this is to block that person.

Q: Can I delete my messages?

Yes. Previously, you could unsend your messages within 48 hours after sending them. As of Telegram 5.5, you can always delete any messages you sent or received in *any* one-on-one conversation (in groups, it's still your own messages only). You can also clear the entire chat history on both ends. On Telegram, deleted messages do not leave a mark in the chat.



If my calculations are correct, when this baby hits 88 miles per hour… you're gonna see some serious shit. 17:25

Together with privacy settings for forwarded messages, this makes exchanging Telegram messages similar to talking face to face (without a tape recorder). As a result, our users no longer need to worry about the data accumulating in their chats over the years. Both parties in a conversation now have full control over what does and what doesn't belong to their online identity.

Q: Can I make calls via Telegram?

Yes! Voice calls are currently available to users around the world.

Q: How can I use emoticons in Telegram on my iOS device?

We support emoji emoticons. Simply enable the Emoji keyboard in your iOS device's Settings (General - Keyboards - Add New Keyboard… - Emoji). Then switch to that keyboard whenever you're out of words.

Groups and Channels

Q: What makes Telegram groups cool?

Telegram groups can have up to **200,000 members** each and are extremely powerful communication tools. Here are a few key features that make them stand out in the messaging world:

Unified history
Edit your messages after posting, delete them so that they disappear for everyone.

Cross-platform availability
Access your messages anytime, from any number of your mobile or desktop devices.

Instant search
Find the message you're looking for, even among millions. Filter by sender to make searching easier.

Replies, mentions, hashtags
Easily trace a conversation and keep communication efficient, no matter the group size.

Smart notifications
Mute the group to get notifications only when people mention you or reply to your messages.

Pinned messages
You can pin any message to be displayed at the top of the chat screen. All members will get a notification — even if they muted ordinary messages from your group.

Moderation tools
Appoint administrators that can mass-delete messages, control membership, and pin important messages. Define their admin privileges with granular precision.

Group permissions
Set default permissions to restrict all members from posting specific kinds of content. Or even restrict members from sending messages altogether - and let the admins chat amongst themselves while everybody else is watching.

File sharing
Send and receive files of any type, up to 1,5 GB in size each, access them instantly on your other devices.

Public groups
Get a short link for your group and make it public, like t.me/publictestgroup. This way, anybody can view the group's entire chat history and join to post messages.

Customization via bots
Create custom tools for any specific needs using our Bot API and Inline Bots.

Q: What's the difference between groups and channels?

Telegram **groups** are ideal for sharing stuff with friends and family or collaboration in small teams. But groups can also grow very large and support communities of up to 200,000 members. You can make any group **public**, toggle **persistent history** to control whether or not new members have access to earlier messages and appoint **administrators** with granular privileges. You can also pin important messages to the top of the screen so that all members can see them, including those who have just joined.

Channels are a tool for broadcasting messages to large audiences. In fact, a channel can have an unlimited number of subscribers. When you post in a channel, the message is signed with the channel's name and photo and not your own. Each message in a channel has a **view counter** that gets updated when the message is viewed, including its forwarded copies.

> Read more about channels in the Channels FAQ »

Q: How do I create a group?

iOS: Start a new message (tap the icon in the top right corner in Chats) > 'New Group'.
Android: Tap the circular pencil icon in the chat list > 'New Group'.

Android: Tap the circular pencil icon in the chat list > 'New Group'.
Telegram Desktop: Click the menu button in the top left corner > 'New Group'.

Q: Can I assign administrators?

Telegram groups are democratic by design. Everyone can invite new members and change the group's name and photo, which is ideal for small bands of friends or coworkers. For larger communities that need more administration, we've introduced a special restricted mode.

In the restricted mode, only administrators can add new people and change the name and photo of the group. You can define their admin privileges with granular precision – from deleting messages to appointing new administrators.

iOS: Go to Group Info (tap the photo in the top right corner on the group's chat screen) — Edit — Add Admins. Disable 'All Members Are Admins' and appoint admins from the list.
Android: Go to Group Info (tap the name in the header) — ... (in the top right corner) — Set Admins. Disable 'All Members Are Admins' and appoint admins from the list.
Telegram Desktop: When in the group, click '...' in the top right corner > 'Manage group' > 'Manage administrators'. Disable 'All Members Are Admins' and appoint people from the list.

Q: How do I add more members? What's an invite link?

You can add your contacts, or using search by username.

It is easy to migrate existing groups to Telegram by sending people an **invite link**. To create an invite link, go to Group Info > Add Member > Invite to Group via Link.

Anyone who has Telegram installed will be able to join your group by following this link. If you choose to revoke the link, it will stop working immediately.

> Read more about invite links in our blog »

Usernames and t.me

Q: What are usernames? How do I get one?

You can set up a **public** username on Telegram. It then becomes possible for other users to find you by that username — you will appear in contacts search under 'global results'. Please note that people who find you will be able to send you messages, even if they don't know your number. If you are not comfortable with this, we advise against setting up a username in Telegram.

You can set up a username in Settings and use the universal search box in the chat list to search for chats, messages, and usernames.

Q: How does t.me work?

Once you've set up a username, you can give people a t.me/username link. Opening that link on their phone will automatically fire up their Telegram app and open a chat with you. You can share username links with friends, write them on business cards or put them up on your website.

This way people can contact you on Telegram without knowing your phone number.

Q: What can I use as my username?

You can use a-z, 0-9 and underscores. Usernames are case-insensitive, but Telegram will store your capitalization preferences (e.g. Telegram and TeleGram is the same user). The username must be at least five characters long.

Q: Do I need a username?

You don't have to get one. Remember that Telegram usernames are public and choosing a username on Telegram makes it possible for people to find you in global search and send you messages even if they don't have your number. If you are not comfortable with this, we advise against setting up a username.

Q: If someone finds me by username, messages and I reply — will they know my number?

No. Neither party will see another's phone number (unless this is permitted by your privacy settings). This is similar to the case when you message a person who you've met in a Telegram group.

Q: How do I delete my username?

Go to Settings and save an empty username. This will remove your username; people will no longer be able to find you via search. This will not affect existing conversations.

Q: What do I do if my username is taken?

Telegram usernames are distributed on a first come — first serve basis.

We understand that certain usernames are part of an online identity for some of us. If your desired username is already taken, we will be happy to help you acquire it for your account or channel, provided that you have that same username on at least **two** of these services: Facebook, Twitter, Instagram.

Due to the fact that one account can register multiple bot and channel usernames, we reserve the right to recall usernames assigned to unused bots and channels, as well as openly squatted usernames.

To request a username, contact @Username_bot.

Q: What if someone is pretending to be me?

If a scammer is pretending to be you, please contact @NoToScam.

Security

> If you are an advanced user, you may find our FAQ for the Technically Inclined useful as well.

Q: How secure is Telegram?

Telegram is more secure than mass market messengers like WhatsApp and Line. We are based on the MTProto protocol (see description and advanced FAQ), built upon time-tested algorithms to make security compatible with high-speed delivery and reliability on weak connections. We are continuously working with the community to improve the security of our protocol and clients.

Q: What if I'm more paranoid than your regular user?

We've got you covered. Telegram's special secret chats use end-to-end encryption, leave no trace on our servers, support self-destructing messages and don't allow forwarding. On top of this, secret chats are not part of the Telegram cloud and can only be accessed on their devices of origin.

Q: So how do you encrypt data?

We support two layers of secure encryption. Server-client encryption is used in Cloud Chats (private and group chats), Secret Chats use an additional layer of client-client encryption. All data, regardless of type, is encrypted in the same way — be it text, media or files.

Our encryption is based on 256-bit symmetric AES encryption, 2048-bit RSA encryption, and Diffie–Hellman secure key exchange. You can find more info in the Advanced FAQ.

> See also: Do you process data requests?

Q: Why should I trust you?

Telegram is open, anyone can check our source code, protocol and API, see how everything works and make an informed decision. In fact, we welcome security experts to audit our system and will appreciate any feedback at security@telegram.org (more about this)

On top of that, Telegram's primary focus is not to bring a profit, so commercial interests will never interfere with our mission.

Q: Do I need to trust Telegram for this to be secure?

When it comes to secret chats, you don't — just make sure that the visualized key of your secret chat matches the one in your friend's secret chat settings. More about this below.

Q: What if my hacker friend says they could decipher Telegram messages?

Anyone who claims that Telegram messages can be deciphered is welcome to prove that claim in our competition and win $300,000. You can check out the Cracking Contest Description to learn more.

Any comments on Telegram's security are welcome at security@telegram.org. All submissions which result in a change of code or configuration are eligible for bounties, ranging from **$500** to **$100,000** or more, depending on the severity of the issue. Please note that we can not offer bounties for issues that are disclosed to the public before they are fixed.

Q: Can Telegram protect me against everything?

Telegram can help when it comes to data transfer and secure communication. This means that all data (including media and files) that you send and receive via Telegram cannot be deciphered when intercepted by your internet service provider, owners of Wi-Fi routers you connect to, or other third parties.

But please remember that we cannot protect you from your own mother if she takes your unlocked phone without a passcode. Or from your IT-department if they access your computer at work. Or from any other people that get physical or root access to your phones or computers running Telegram.

If you have reasons to worry about your personal security, we strongly recommend using only Secret Chats in official or at least verifiable open-source apps for sensitive information, preferably with a self-destruct timer. We also recommend enabling 2-Step Verification and setting up a strong passcode to lock your app, you will find both options in Settings — Privacy and Security.

Q: How does 2-Step Verification work?

Logging in with an SMS code is an industry standard in messaging, but if you're looking for more security or have reasons to doubt your mobile carrier or government, we recommend protecting your cloud chats with an additional password.

You can do this in **Settings** – **Privacy and Security** – **2-Step Verification**. Once enabled, you will need both an SMS code and a password to log in. You can also set up a recovery email address that will help regain access, should you forget your password. If you do so, please remember that it's important that the recovery email account is also protected with a strong password and 2-Step Verification when possible.

Check this out for tips on creating a strong password that is easy to remember.

Q: Why can jailbroken and rooted devices be dangerous?

Using a rooted or jailbroken device makes it easier for a potential attacker to gain full administrative control over your device — root access.

A user with root access can easily bypass security features built into the operating system, read process memory or access restricted areas, such as the internal storage. Once an attacker has root access, any efforts to mitigate threats become futile. No application can be called safe under these circumstances, no matter how strong the encryption.

Secret Chats

Q: How are secret chats different?

Secret chats are meant for people who want more secrecy than the average fella. All messages in secret chats use end-to-end encryption. This means only you and the recipient can read those messages — nobody else can decipher them, including us here at Telegram (more on this here). On top of this, Messages cannot be forwarded from secret chats. And when you delete messages on your side of the conversation, the app on the other side of the secret chat will be ordered to delete them as well.

You can order your messages, photos, videos and files to self-destruct in a set amount of time after they have been read or opened by the recipient. The message will then disappear from both your and your friend's devices.

All secret chats in Telegram are device-specific and are not part of the Telegram cloud. This means you can only access messages in a secret chat from their device of origin. They are safe for as long as your device is safe in your pocket.

Q: How do I start a secret chat?

iOS: Start a new message (tap the icon in the top-right corner in Messages). Then 'New secret chat'.
Android: Swipe right to open the menu, then 'New secret chat'.

Remember that Telegram secret chats are device-specific. If you start a secret chat with a friend on one of your devices, this chat will only be available on that device. If you log out, you will lose all your secret chats. You can create as many different secret chats with the same contact as you like.

Q: How do self-destructing messages work?

The Self-Destruct Timer is available for **all messages** in Secret Chats and for **media** in private cloud chats.

To set the timer, simply tap the clock icon (in the input field on iOS, top bar on Android), and then choose the desired time limit. The clock starts ticking the moment the message is displayed on the recipient's screen (gets two green checks). As soon as the time runs out, the message disappears from **both** devices.

Media sent with short self-destruct timers (< 1 minute) can only be viewed while you're holding your finger on them — and we will try to send a notification whenever a screenshot is taken.

Please note that the timer in Secret Chats only applies to messages that were sent **after** the timer was set. It has no effect on earlier messages.

Q: Can I be certain that my conversation partner doesn't take a screenshot?

Unfortunately, there is no bulletproof way of detecting screenshots on certain systems (most notably, some Android and Windows Phone devices). We will make every effort to alert you about screenshots taken in your Secret Chats, but it may still be possible to bypass such notifications and take screenshots silently. We advise to share sensitive information only with people you trust. After all, nobody can stop a person from taking a picture of their screen with a different device or an old school camera.

Q: What is this 'Encryption Key' thing?

When a secret chat is created, the participating devices exchange encryption keys using the so-called Diffie-Hellman key exchange. After the secure end-to-end connection has been established, we generate a picture that visualizes the encryption key for your chat. You can then compare this image with the one your friend has — if the two images are the same, you can be sure that the secret chat is secure, and no man-in-the-middle attack can succeed.

Newer versions of Telegram apps will show a larger picture along with a textual representation of the key (this is not the key itself, of course!) when both participants are using an updated app.

Always compare visualizations using a channel that is known to be secure — it's safest if you do this in person, in an offline meeting with the conversation partner.

Q: Why not just make all chats 'secret'?

All Telegram messages are always securely encrypted. Messages in Secret Chats use **client-client** encryption, while Cloud Chats use **client-server/server-client** encryption and are stored encrypted in the Telegram Cloud (more here). This enables your cloud messages to be both secure and immediately accessible from any of your devices - even if you lose your device altogether.

The problem of restoring access to your chat history on a newly connected device (e.g. when you lose your phone) does not have an elegant solution in the end-to-end encryption paradigm. At the same time, reliable backups are an essential feature for any mass-market messenger. To solve this problem, some applications (like Whatsapp and Viber) allow decryptable backups that put their users' privacy at risk - even if they do not enable backups themselves. Other apps ignore the need for backups altogether and fade into oblivion before ever reaching a million users.

We opted for a third approach by offering two distinct types of chats. Telegram disables default system backups and provides all users with an integrated security-focused backup solution in the form of Cloud Chats. Meanwhile, the separate entity of Secret Chats gives you full control over the data you do not want to be stored.

This allows Telegram to be widely adopted in broad circles, not just by activists and dissidents, so that the simple fact of using Telegram does not mark users as targets for heightened surveillance in certain countries. We are convinced that the separation of conversations into Cloud and Secret chats represents the most secure solution currently possible for a massively popular messaging application.

> See also: Why Telegram isn't End-to-End Encrypted "by Default"

Your Account

Q: Who can see my phone number?

On Telegram, you can send messages in private chats and groups without making your phone number visible. By default, your number is only visible to people who you've added to your address book as contacts. You can further modify this in *Settings > Privacy and Security > Phone Number.*

> Note that people will always see your number if they know it already and **saved** it in their address book.

Q: I have a new phone number, what do I do?

Each phone number is a **separate** account on Telegram. You have several options if you are using multiple phone numbers:

- If you will **no longer use the old number** (e.g., you moved to a new country or changed your number for good), simply go to Settings and change the number connected to your Telegram account to the new number. **Important**: make sure you have access to your connected phone number - otherwise you risk losing access to your account.
- If you will use the new number for a **limited time** (e.g., you're on a trip or vacation), there's no need to do anything.
- If you want to keep using **both numbers** (e.g., you have a work phone and personal phone), choose one as your Telegram number. You *may* create another Telegram account on the second number as well, for example, if you want to keep work and personal chats separated. It is possible to log in to one Telegram app with up to **4** different accounts at once.

Q: How do I log out?

Most users don't need to log out of Telegram:

- You can use Telegram on many devices **at the same time**. Just use the same phone number to log in on all devices.
- You can go to *Settings > Data and Storage > Storage Usage> Clear cache* to **free up space** on your device without logging out.
- If you use Telegram with **multiple phone numbers**, you can switch between accounts without logging out.
- If you use Telegram on a **shared device**, you can set up a passcode in *Settings > Privacy and Security* to make sure only you have access to your account.

If you do want to log out for some reason, here's how you do that:

iOS: Go to Settings — Edit — Log out.
Android, *Telegram Desktop*: Go to Settings — ... (in the top right corner) — Log out.

If you log out, you will keep all your cloud messages. However, you **will lose** all your **Secret Chats** and **all messages** inside those secret chats when you log out.

> Note that logging out does **not** trigger remote deletion of your secret chat messages on your partner's device — to do that, choose 'Clear History' first.

Q: How do I change my phone number?

You can change your number in Telegram and keep **everything**, including all your contacts, messages, and media from the Telegram cloud, as well as all your Secret Chats on all devices.

To change your number, go to Settings, then tap on your phone number (just above the username), then 'Change Number'. If you already have a different Telegram account on the target number, you'll need to delete that account first.

Q: How do I delete my account?

If you would like to delete your account, you can do this on the deactivation page. Deleting your account permanently removes all your **messages** and **contacts**. All groups and channels that you've created are orphaned and left without a creator but admins retain their rights.

This action must be confirmed via your Telegram account and cannot be undone.

> We recommend using a non-mobile browser for this process.
> Note that you'll receive the **code** via **Telegram**, not SMS.

Q: What happens if I delete my account?

As was just mentioned above, all your data will be flushed from our system: all messages, groups, and contacts associated with your account will be deleted. That said, your contacts will still be able to chat in the groups that you have created, and they will still have *their* copy of the messages you sent them. So if you want to send messages that can vanish without a trace, try using our self-destruct timer instead.

Termination of a Telegram account is irreversible. If you sign up again, you will appear as a new user and will not get your history, contacts or groups back. People, who have your phone number in their contacts, will be notified. The new user will be displayed as a separate conversation in their messages list and their conversation history with this new user will be empty.

Q: How does account self-destruction work?

Telegram is not a commercial organization, and we value our disk space greatly. If you stop using Telegram and don't come online for at least six months, your account will be deleted along with all messages, media, contacts and every other piece of data you store in the Telegram cloud. You can change the exact period after which your inactive account will self-destruct in Settings.

Q: My phone was stolen, what do I do?

First of all, sorry about your phone. Unfortunately, the phone number is the only way for us to identify a Telegram user at the moment. We don't collect additional information about you, so whoever has the number, has the account. This means we can't help you unless you have access either to the phone number or to Telegram itself on any of your devices.

I have access to Telegram on another device

1. Go to Telegram Settings — Privacy and Security and turn on Two-Step Verification. This way the phone number alone will not be enough to log in to your account.
2. Go to Settings — Privacy and Security — Active Sessions and terminate your Telegram session on the old device. Whoever has your phone will not be able to log in again, since they don't know your password.
3. Contact your phone provider, so that they block your old SIM and issue a new one with your number.
4. If you decide to switch to a new phone number, don't forget to go to Settings, tap on your phone number and change your Telegram number to the new one.

I don't have access to Telegram on any other devices

1. First and foremost, you need to contact your phone provider, so that they block your old SIM and issue a new one with your number.
2. Wait till you receive your new SIM with the old number, log in to Telegram, then go to Settings — Privacy and Security — Active Sessions and terminate your Telegram session on the old device.

Removing sensitive data

Common thieves usually throw out the SIM card immediately (the phone is harder to locate this way), then wipe the devices and sell them, so there isn't much risk for the data in case of regular petty theft. But if you have reasons to worry about the data on the device and are unable to log out the other device, it is best that you wipe it remotely. You can read more about it here: Apple iOS, Android. Unfortunately, this requires you to have prepared in advance for this scenario.

You can delete your Telegram account if you are logged in on at least one of your other devices (mobile or desktop). Note that inactive Telegram accounts self-destruct automatically after a period of time — 6 months being the default setting.

Bots

> If you're a developer, you may find our Bots FAQ more useful.

Q: What are bots?

Bots are like small programs that run right inside Telegram. They are made by third-party developers using the Telegram Bot API.

Q: How do I create a bot?

Creating Telegram bots is super-easy, but you will need at least some skills in computer programming. If you're sure you're up to it, our **Introduction for Developers** is a good place to start.

Unfortunately, there are no out-of-the-box ways to create a working bot if you are not a developer. But we're sure you'll soon find plenty of bots created by other people to play with.

Q: A bot is sending me messages, how do I make it stop?

If you don't want a bot to send you messages, feel free to block it - same as you would block a human user. Some Telegram clients have a 'Stop Bot' button right in the bot's profile.

That said, most bot developers offer commands that silence the bot, check its */help* for clues.

Q: Are bots safe?

Yes. Bots are no different from human users that you meet in groups for example. They can see your public name, username, and profile pictures, and they can see messages you send to them, that's it. They can't access your last seen status and **don't** see your phone number (unless you decide to give it to them yourself).

Naturally, any bot should be treated as a stranger — don't give them your passwords, Telegram codes or bank account numbers, even if they ask nicely. Also, be careful when opening files sent by bots, same as you would deal with ordinary humans. Example: If a bot sent us a file called *OpenMe.exe*, we probably wouldn't open it.

Q: If I add a bot to my group, can it read my messages?

Bots can work in two modes when you add them to groups. By default, bots only see messages that are meant for them. In this case, you'll see 'has no access to messages' in the group members list next to the bot.

Some bots need more information to work, so developers may disable the privacy mode. In this case, the bot will see all messages sent to the group, and you will see 'has access to messages' in the members list next to the bot.

Learn more about privacy mode for bots »

If your group contains very sensitive information, maybe it's better to avoid adding bots you don't trust 100%.

Q: Are bots made by Telegram?

No. While we have some official bots for specific purposes (like @gif or @GDPRbot), we don't usually make bots. Bots are made by third-party developers using the Telegram Bot API and platform.

Q: Where can I find more bots?

There is no official store at the moment, so you'll have to ask your friends or search the web for now. We're pretty sure you'll find some bots to play with.

Deeper questions

Q: Why not open source everything?

All code will be released eventually. We started with the most useful parts — a well-documented API that allows developers to build new Telegram apps, and open source clients that can be verified by security specialists.

Q: Can I run Telegram using my own server?

Our architecture does not support federation yet. Telegram is a unified cloud service, so creating forks where two users might end up on two different Telegram clouds is unacceptable. To enable you to run your own Telegram server while retaining both speed and security is a task in itself. At the moment, we are undecided on whether or not Telegram should go in this direction.

Q: Can I use the Telegram API?

Yes. Developers for all platforms are welcome to use our protocol, API and even source code. Check out the Getting started section of the docs.

> Don't forget about our Bot API that lets you build cool stuff on our platform.

Q: Do you have a Privacy Policy?

Sure. Check this out.

Q: Why do you have two apps in the Mac App store?

One is our native macOS app, the other is the macOS version of our multi-platform client— Telegram Desktop. Both apps are official. Both started out as unofficial applications by two different developers and vary in design and functionality.

Telegram Desktop is currently the more **stable** version. It is optimized for photo and document sharing, as well as sending long text messages.

The macOS app features native macOS design and supports secret chats.

Q: Can I translate Telegram?

Telegram is officially available in English, Spanish, German, Dutch, Italian, French, Arabic, Portuguese, Korean, Malay, Russian and Ukrainian on most platforms, and we are gradually expanding the list of languages built into the apps.

If you don't like how a specific element in Telegram's interface is translated in your language, or would like to help us maintain the translation, check out our localization platform. Everyone can suggest translations and vote for the best ones, making Telegram localization a community-driven effort.

If you're looking to go beyond suggestions for individual phrases and would like to help us maintain the official translation to your language on a continuous basis, you can contact @TelegramAuditions. Please include a hashtag with the English name of your language (e.g. #Albanian) and a few links to phrases on this platform with your **translation suggestions** or **comments**. Be sure read the **Style Guide** carefuly before you apply.

Q: Can I help?

Yes, we are always looking for volunteers to help us with user support. If you would be interested in answering questions about Telegram to users from your country, contact our auditions account.

Before you apply, please check out the Telegram Support Initiative.

Passport

Telegram Passport is a unified authorization method for services that require personal identification. With Telegram Passport, you can upload your documents once, then instantly share your data with services that require real-world ID (finance, ICOs, etc.).

Your identity documents and personal data will be stored in the Telegram cloud using **End-to-End Encryption**. To Telegram, this data is just random gibberish, and we have no access to the information you store in your Telegram Passport. When you share data, it goes directly to the recipient.

> You can find more information about Telegram Passport on our blog.

If you're a developer or owner of a service that requires real-life ID, kindly take a look at **this manual**. You can also try requesting Telegram Passport data using this page.

Troubleshooting

Login and SMS

Please make sure you are entering your mobile phone number in the international format.
I.e.: `+(country code)(city or carrier code)(your number)`

If you are having registration or login problems, please contact us using this form.

Getting a code via a phone call

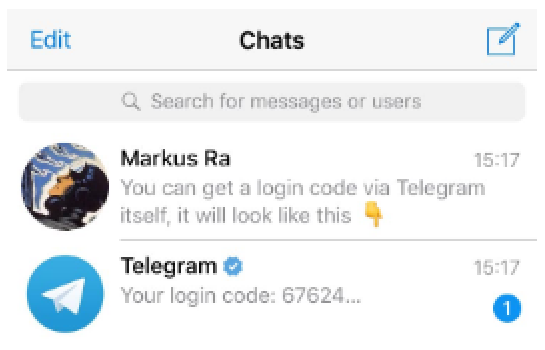
For security reasons, login codes dictated via a phone call are only available for accounts that have **two-step verification** enabled (Settings > Privacy & Security > Two-Step Verification).

Please also note that Telegram accounts can only be connected to a mobile number. We currently don't support landline numbers.

Getting a code via Telegram

If you have recently used one of our apps on **another device** (it could also be a different app on the same device), we may send the login code **via Telegram** instead of SMS.

To receive such a code, just check Telegram from any of your connected devices. You will find it in the chat with Telegram, a verified profile with a blue check:



Markus Ra 15:17
You can get a login code via Telegram itself, it will look like this 👇

Telegram 15:17
Your login code: 67624...

Login code sent via Telegram

WARNING! Please note that getting codes via Telegram should not be considered an alternative to using an **up-to-date phone number**. In case of a change in numbers, always make sure Telegram is connected to a phone number **you control**, otherwise you risk losing access to your account forever.

Notification problems

> If the tips below don't help, check out this detailed guide on **Troubleshooting Notification Issues**.

ANDROID

1. Go to Telegram Settings — Notifications and Sounds, make sure that notifications are **ON** and Importance is set to **"High"** or greater.
2. Check whether contact or group is *muted*.
3. Make sure Google Play Services are installed on your phone.
4. Check **notification priority** for Telegram in **Android settings**, it can be called *Importance* or *Behaviour* depending on your device.
5. If your phone uses some **battery saving software**, make sure that Telegram is whitelisted in that application.

> NOTE: **Huawei** and **Xiaomi** devices have evil task killer services that interfere with the Telegram notification service. For our notifications to work, you need to add Telegram to allowed apps in those devices' security settings. Huawei: Phone Manager App > Protected Apps > Add Telegram to the list. Xiaomi: Services > Security > Permissions > Autostart, find Telegram and enable autostart.

iOS

1. Go to Telegram Settings — Notifications and Sounds, make sure that notifications are ON in Telegram.
2. Check that notifications are **ON** in phone Settings.
3. Check, whether contact or group is *muted*.
4. Shut down Telegram (go to home screen, double tap home button, kill Telegram (swipe upwards), then go to phone settings, set the alert style for Telegram to NONE. Relaunch Telegram, go to phone settings, set alert style back to banners.

Problems with contacts

If you know your friends have Telegram, but you can't see them — or they appear as numbers instead of names.

Android:

1. Make sure you are using the latest version of the app.
2. Relaunch the app (by terminating it from processes list and launching again).
3. Temporarily change the name of the contact in phone contacts (add a few symbols, then change back again).
4. If that didn't help, try to re-login.

iOS:

1. Force quit the app (double tap home button, then swipe up on Telegram), then relaunch and check if it helped.
2. If that doesn't help, temporarily change the name of the contact in phone contacts (add a few symbols, then change back again).
3. If that doesn't work, re-login: Settings - Edit - Log Out. Remember that logging out kills all your Secret Chats. Then log in again.

Deleting contacts on Android

To delete a contact, open a chat with the person, tap on their profile photo in the top area of the chat screen, then tap on '...' in the top right corner — 'Delete'.

If you want to delete the contact completely, make sure you also delete them from your phone contacts. Telegram stays in sync and will add the contact back if you don't.

Where did my Secret Chat messages go?

Secret Chats are established between the two devices they were created on. This means that all those messages are not available in the cloud and cannot be accessed on other devices.

Moreover, Secret Chats are also tied to your current login session on the device. If you log out and in again, you will lose all your Secret Chats.

Can't send messages to non-contacts

When users report unwanted messages from a Telegram account, we apply a limit: Reported accounts can only send messages to people who have their number saved as a contact.

This means that if you randomly contact people you don't know and send them annoying messages, you may lose

This means that if you randomly contact people you don't know and send them annoying messages, you may lose the ability to do so in the future.

If you think that this limit was applied to your account wrongly, please visit this page.

Telegram Support

If you have any other questions, please contact Telegram Support (in Telegram go to Settings — Ask a question). Note that we rely on volunteers for support.

If you can't log in to your account, please use this form.

Twitter?

Yep. Follow us! **@telegram**
Our twitter account in Spanish: @telegram_es
In Italian: @telegram_it
In Korean: @Telegram_kr
In German: @de_telegram
For users from Brazil: @Telegram_br
Our Arabic-speaking users may find @telegram_arabic more interesting.

@SmsTelegram, login help on Twitter

We have a special account that can help you with login problems, **@smstelegram**. This account is official. Don't be afraid to DM it the number you use for Telegram, we need this info to investigate issues.

Be careful, we don't have any other support accounts on any social media platforms.

Facebook?

Facebook seems to have taken down our page for no apparent reason several months ago. Well, who cares, we weren't using it much anyway. (This was the address, in case you're curious: https://www.facebook.com/tlgrm). Right now, if anyone on Facebook is telling you they're us, they are **not**.

Telegram
Telegram is a cloud-based mobile and desktop messaging app with a focus on security and speed.

About
FAQ
Blog
Jobs

Mobile Apps
iPhone/iPad
Android
Windows Phone

Desktop Apps
PC/Mac/Linux
macOS
Web-browser

Platform
API
Translations
Instant View

Telegram Applications

On this page you can find **Telegram apps** for every platform and links to their **open source** code.

Mobile apps

Telegram for Android
Telegram for iPhone and iPad
Telegram for WP

Desktop apps

Telegram for Windows/Mac/Linux
Telegram for macOS

Web apps

Telegram Web-version
Telegram Chrome app

Telegram Database Library (TDLib)

TDLib – a cross-platform client designed to facilitate creating custom apps on the Telegram platform.
Telegram X for Android – a slick experimental Telegram client based on TDLib.

Unofficial apps

Unigram, a client optimized for Windows 10 (based on TDLib) (desktop and Xbox One)
Telegram CLI for Linux
MadelineProto

Source code

For the moment we are focusing on open sourcing the things that allow developers to quickly build something using our API. We have published the code for our Android, iOS, web and desktop apps (Win, macOS and Linux) as well as the Telegram Database Library.

> This code allows security researchers to **fully evaluate** our end-to-end encryption implementation.

Telegram Database Library
Cross-platform library for building custom Telegram apps, see TDLib for details.
Licensed under Boost 1.0.
GitHub »

Telegram for Android
Official Android App, see Google Play Market page for full description.
Licensed under GNU GPL v. 2 or later.
GitHub »

Telegram for iOS
Licensed under GNU GPL v. 2 or later.
GitHub »

Telegram for macOS
Native macOS client.
Licensed under GNU GPL v. 2.
GitHub »

Telegram for Web browsers
Javascript client for browsers. Mac, Windows, Linux.
Licensed under GNU GPL v. 3.
GitHub »

Telegram Desktop
Qt-based desktop client. Mac, Windows, Linux.
Licensed under GNU GPL v. 3.
GitHub »

Telegram for WP
Licensed under GNU GPL v. 2 or later.
Available here »

Unofficial apps

Telegram CLI (Unofficial)
Linux Command-line interface for Telegram.
Licensed under GNU GPL v. 2.
GitHub »

Unigram (Unofficial)
A Telegram client optimized for Windows 10 (desktop and Xbox One).
Licensed under GNU GPL v. 3 or later.
GitHub »

MadelineProto (Unofficial)
A PHP MTProto Telegram client.
Licensed under GNU AGPL v. 3
GitHub »

Contact for security researchers

If you find any issues with Telegram apps and protocol, or have any questions about our implementation of security features, kindly contact us at **security@telegram.org**.

Telegram APIs

We offer two kinds of APIs for developers. The **Bot API** allows you to easily create programs that use Telegram messages for an interface. The **Telegram API and TDLib** allow you to build your own customized Telegram clients. You are welcome to use both APIs free of charge.

You can also add **Telegram Widgets** to your website.

Designers are welcome to create **Animated Stickers** or **Custom Themes** for Telegram.

Bot API

This API allows you to connect bots to our system. **Telegram Bots** are special accounts that do not require an additional phone number to set up. These accounts serve as an interface for code running somewhere on your server.

To use this, you don't need to know anything about how our MTProto encryption protocol works — our intermediary server will handle all encryption and communication with the Telegram API for you. You communicate with this server via a simple HTTPS-interface that offers a simplified version of the Telegram API.

Learn more about the Bot API here »

Bot developers can also make use of our **Payments API** to accept **payments** from Telegram users around the world.

TDLib – build your own Telegram

Even if you're looking for maximum customization, you don't have to create your app from scratch. Try our **Telegram Database Library** (or simply TDLib), a tool for third-party developers that makes it easy to build fast, secure and feature-rich Telegram apps.

TDLib takes care of all **network implementation** details, **encryption** and **local data storage**, so that you can dedicate more time to design, responsive interfaces and beautiful animations.

TDLib supports all Telegram features and makes developing Telegram apps a breeze on any platform. It can be used on Android, iOS, Windows, macOS, Linux and virtually any other system. The library is open source and compatible with virtually **any programming language**.

Learn more about TDLib here »

Telegram API

This API allows you to build your own customized Telegram clients. It is 100% open for all developers who wish to create Telegram applications on our platform. Feel free to study the open source code of existing Telegram applications for examples of how things work here. Don't forget to register your application in our system.

- Getting Started
- Security
- Optimization
- API methods

Getting started

Creating an application

How to get your application identifier and create a new Telegram app.

User authorization

How to register a user's phone to start using the API.

Two-factor authentication

How to login to a user's account if they have enabled 2FA, how to change password.

Error handling

How to handle API return errors correctly.

Handling different data centers

How to connect to the closest DC access point for faster interaction with the API, and things to watch out for when developing a client.

Handling updates

How to subscribe to updates and handle them properly.

Handling PUSH-notifications

How to subscribe and handle them properly.

Channels, supergroups and groups

How to handle channels, supergroups, groups, and what's the difference between them.

Calling methods

Additional options for calling methods.

Uploading and Downloading Files

How to transfer large data batches correctly.

Pagination

How to fetch results from large lists of objects.

Security

Secret chats, end-to-end encryption

New feature for end-to-end-encrypted messaging.

Security guidelines

Important checks required in your client application.

Perfect Forward Secrecy

Binding temporary authorization key to permanent ones.

Optimization

Client optimization
Ways to boost API interactions.

API methods

Available method list
A list of available high-level methods.

API TL-schema, as JSON
Text and JSON-presentation of types and methods used in API.

Available layer list
A list of available schema versions.

Other articles

Admin, banned and default rights for channels, supergroups and groups
How to handle admin permissions, granular bans and global permissions in channels, groups and supergroups.

Min constructors
Sometimes, user and channel constructors met in group chat updates may not contain full info about the user: how to handle such constructors.

Account deletion
How to reset an account if the 2FA password was forgotten.

Telegram Passport
How to work with Telegram Passport directly using the MTProto API.

Telegram Payments
How to work with Telegram Payments directly using the MTProto API.

Styled text with message entities
How to create styled text with message entities

Message drafts
How to handle message drafts

Top peer rating
If enabled, the rating of top peers indicates the relevance of a frequently used peer in a certain category (frequently messaged users, frequently used bots, inline bots, frequently visited channels and so on).

Handling file references
How to handle file references.

Seamless Telegram Login
Handle Seamless Telegram Login URL authorization requests.

Web events
When interacting with HTML5 games and the websites of payment gateways, Telegram apps should expose the following JS APIs.

Creating your Telegram Application

We welcome all developers to use our API and source code to create Telegram-like messaging applications on our platform free of charge.

> In order to ensure consistency and security across the Telegram ecosystem,
> **all third-party client apps** must comply with the **API Terms of Service**.

Obtaining api_id

In order to obtain an **API id** and develop your own application using the Telegram API you need to do the following:

- Sign up for Telegram using any application.
- Log in to your Telegram core: https://my.telegram.org.
- Go to 'API development tools' and fill out the form.
- You will get basic addresses as well as the **api_id** and **api_hash** parameters required for user authorization.
- For the moment each number can only have one api_id connected to it.

We will be sending important developer notifications to the phone number that you use in this process, so please use an up-to-date number connected to your active Telegram account.

Using Telegram's open source code

Everyone is welcome to use our open source code. We have included a sample API id with the code. This API id is limited on the server side and is not suitable for apps released to end-users — using it for anything but testing purposes will result in the API_ID_PUBLISHED_FLOOD error for your users. It is necessary that you obtain your **own API id** before you publish your app.

> Please remember to publish your code as well in order to comply with the GNU GPL licences.

User Authorization

Authorization is associated with a client's encryption key identifier: **auth_key_id**. No additional parameters need to be passed into methods following authorization.

Sending a verification code

Example implementations: telegram for android, tdlib.

Authorization requires that a text message containing an authorization code first be sent to the user's phone.
This may be done using the auth.sendCode method.
The system will automatically choose how to send the authorization code; there are four possible ways the code can arrive:

- Telegram code
- SMS code
- Phone call: a synthesized voice will tell the user which verification code to input
- Flash phone call: the code will be sent via a flash phone call, that will be closed immediately.
  In the last case, the phone code will then be the phone number itself, just make sure that the phone number matches the specified pattern (see auth.sentCodeTypeFlashCall).

The auth.sendCode method also has parameters for enabling/disabling use of flash calls, and allows passing an SMS token that will be included in the sent SMS.
For example, the latter is required in newer versions of android, to use the android SMS receiver APIs.

The returned auth.SentCode object will contain multiple parameters:

flags	#	Flags, see TL conditional fields
type	auth.SentCodeType	Phone code type
phone_code_hash	string	Phone code hash, to be stored and later re-used with auth.signIn
next_type	flags.1? auth.CodeType	Phone code type that will be sent next, if the phone code is not received within `timeout` seconds: to send it use auth.resendCode
timeout	flags.2?int	Timeout for reception of the phone code

If the message takes too long (`timeout` seconds) to arrive at the phone, the auth.resendCode method may be invoked to resend a code of type `next_type`.
If the same happens again, you can use auth.resendCode with the `next_type` returned by the previous call to auth.resendCode.
To cancel the verification code use auth.cancelCode.

Sign in/sign up

When user enters verification code, the auth.signIn method must be used to validate it and possibly sign user in.

If the code was entered correctly, but the method returns auth.authorizationSignUpRequired, it means that account with this phone number doesn't exist yet: user needs to provide basic information, accept terms of service and then the new user registration method (auth.signUp) must be invoked.

2FA

When trying to sign in using auth.signIn, an error 400 SESSION_PASSWORD_NEEDED may be returned, if the user has two-factor authentication enabled.
In this case, instructions for SRP 2FA authentication must be followed.

To set up two-factor authorization on an already authorized account, follow the SRP 2FA authentication docs.

We are authorized

As a result of authorization, the client key, **auth_key_id**, becomes associated with the user, and each subsequent API call with this key will be executed with that user's identity. The authorization method itself returns the relevant user. It is best to immediately store the User ID locally in a binding with the key.

Only a small portion of the API methods are available to **unauthorized** users:

- auth.sendCode
- auth.resendCode
- account.getPassword
- auth.checkPassword
- auth.checkPhone
- auth.signUp
- auth.signIn
- auth.importAuthorization
- help.getConfig
- help.getNearestDc
- help.getAppUpdate
- help.getCdnConfig
- langpack.getLangPack
- langpack.getStrings
- langpack.getDifference
- langpack.getLanguages
- langpack.getLanguage

Other methods will result in an error: **401 UNAUTHORIZED**.

Two-factor authentication

Telegram uses the Secure Remote Password protocol version 6a to implement 2FA.

Example impementation: tdlib.

Checking the password with SRP

To login to an account protected by a 2FA password or to perform some other actions (like changing channel owner), you will need to verify user's knowledge of current 2FA account password.

To do this, first the client needs to obtain SRP parameters and the KDF algorithm to use to check the validity of the password via account.getPassword method. For now, only the passwordKdfAlgoSHA256SHA256PBKDF2HMACSHA512iter100000SHA256ModPow algorithm is supported, so we'll only explain that.

Then, after the user provides a password, the client should generate InputCheckPasswordSRP object using SRP and a specific KDF algorithm as shown below and pass it to appropriate method (e.g. auth.checkPassword in case of authorization).

This extension of the SRP protocol uses the password-based PBKDF2 with the 100000 iterations using sha512 (`PBKDF2HMACSHA512iter100000`).
PBKDF2 is used to additionally rehash the `x` parameter, obtained using a method similar to the one described in RFC 2945 (`H(s | H ( I | password | I) | s)` instead of `H(s | H ( I | ":" | password)`) (see below).

Here, `|` denotes concatenation and `+` denotes the arithmetical operator `+`.
In all cases where concatenation of numbers passed to hashing functions is done, the numbers must be used in big-endian form, padded to 2048 bits; all maths is modulo `p`.
Instead of `I`, `salt1` will be used (see SRP protocol).
Instead of `s`, `salt2` will be used (see SRP protocol).

The main hashing function `H` is sha256:

- `H(data) := sha256(data)`

The salting hashing function `SH` is defined as follows:

- `SH(data, salt) := H(salt | data | salt)`

The primary password hashing function is defined as follows:

- `PH1(password, salt1, salt2) := SH(SH(password, salt1), salt2)`

The secondary password hashing function is defined as follows:

- `PH2(password, salt1, salt2) := SH(pbkdf2(sha512, PH1(password, salt1, salt2), salt1, 100000), salt2)`

Client-side, the following parameters are extracted from the passwordKdfAlgoSHA256SHA256PBKDF2HMACSHA512iter100000SHA256ModPow object, contained in the account.password object.

- `g := algo.g`
- `p := algo.p`
  The client is expected to check whether **p** is a safe 2048-bit prime (meaning that both **p** and **(p-1)/2** are prime, and that `2^2047 < p < 2^2048`), and that **g** generates a cyclic subgroup of prime order **(p-1)/2**, i.e. is a quadratic residue **mod p**. Since **g** is always equal to 2, 3, 4, 5, 6 or 7, this is easily done using quadratic reciprocity law, yielding a simple condition on **p mod 4g** — namely, **p mod 8 = 7** for **g = 2**; **p mod 3 = 2** for **g = 3**; no extra condition for **g = 4**; **p mod 5 = 1 or 4** for **g = 5**; **p mod 24 = 19 or 23** for **g = 6**; and **p mod 7 = 3, 5 or 6** for **g = 7**. After **g** and **p** have been checked by the client, it makes sense to cache the result, so as to avoid repeating lengthy computations in future. This cache might be shared with one used for Authorization Key generation.

  If the client has an inadequate random number generator, it makes sense to use the **secure_random** of account.password as additional seed.

- `password := (user-provided password)`

- `salt1 := algo.salt1`
- `salt2 := algo.salt2`
- `g_b := srp_B`
  `srp_B` and `srp_id` are extracted from the account.password object.

The `k` parameter is generated, both on client and server:

- `k := H(p | g)`

The shared param `u` is generated: the client does this, and the server does the same with the `g_a` we will send him later (see below)

- `u := H(g_a | g_b)`

The final parameters are generated client-side only:

- `x := PH2(password, salt1, salt2)`
- `v := pow(g, x) mod p`

The server already has `v`, from when we set the password.

A final shared param is generated, for commodity:

- `k_v := (k * v) mod p`

Finally, the key exchange process starts on both parties.

The client computes a 2048-bit number **a** (using sufficient entropy or the server's **random**; see above) and generates:

- `g_a := pow(g, a) mod p`.

The server computes a 2048-bit number **b** using sufficient entropy and generates the `g_b` parameter that was sent to us (see above).

- `g_b := (k_v + (pow(g, b) mod p)) mod p`

Finally, the SRP session keys are generated:

Client side:

- `t := (g_b - k_v) mod p` (positive modulo, if the result is negative increment by `p`)
- `s_a := pow(t, a + u * x) mod p`
- `k_a := H(s_a)`

Server side:

- `s_b := pow(g_a * (pow(v, u) mod p), b) mod p`
- `k_b := H(s_b)`

Since:

- `g_b := (k_v + (pow(g, b) mod p)) mod p`
- `t := (g_b - k_v) mod p`
- `t := ((k_v + (pow(g, b) mod p)) - k_v) mod p`
- `t := pow(g, b) mod p`
- `s_a := pow(t, a + u * x) mod p`
- `s_a := pow(pow(g, b) mod p, a + u * x) mod p`

And:

- `g_a := pow(g, a) mod p`
- `v := pow(g, x) mod p`

- `v := pow(g, x) mod p`
- `s_b := pow(g_a * (pow(v, u) mod p), b) mod p`
- `s_b := pow((pow(g, a) mod p) * (pow(pow(g, x) mod p, u) mod p), b) mod p`
- `s_b := pow(pow(g, a + x * u) mod p, b) mod p`
- `s_b := pow(pow(g, b) mod p, a + u * x) mod p`
- `s_a := pow(pow(g, b) mod p, a + u * x) mod p`

This means:

- `s_b === s_a`
- `k_b === k_a`

Finally, as per SRP:

- `M1 := H(H(p) xor H(g) | H2(salt1) | H2(salt2) | g_a | g_b | k_a)`

`M1` is passed to inputCheckPasswordSRP, along with `g_a` (as `A` parameter) and the `srp_id`, extracted from the account.password object.

The server then computes:

- `M2 := H(H(p) xor H(g) | H2(salt1) | H2(salt2) | g_a | g_b | k_b)`

Since we said that:

- `s_b === s_a`
- `k_b === k_a`

This means, if everything was done correctly,

- `M1 === M2`

If the password isn't correct, 400 PASSWORD_HASH_INVALID will be returned.

Setting a new 2FA password

To set a new 2FA password use the account.updatePasswordSettings method.
If a password is already set, generate an InputCheckPasswordSRP object as per checking passwords with SRP, and insert it in the `password` field of the account.updatePasswordSettings method.
To remove the current password, do not set any flags in the account.PasswordInputSettings object.

To set a new password, use the SRP parameters and the KDF algorithm obtained using account.getPassword when generating the `password` field.
Then generate a new `new_password_hash` using the KDF algorithm specified in the `new_settings`, just append 32 sufficiently random bytes to the `salt1`, first.
Proceed as for checking passwords with SRP, just stop at the generation of the `v` parameter, and use it as `new_password_hash`:

- `v := pow(g, x) mod p`

As usual in big endian form, padded to 2048 bits.

Email verification

When setting up two-factor authorization, it is recommended to set up a **recovery email**, to allow recovery of the password through the user's email address, in case they forget it.

To set up a recovery email, it must first be verified.
This can be done directly when setting the new password using account.updatePasswordSettings by setting the email parameter and flag in the account.passwordInputSettings constructor.
If the email isn't verified, an EMAIL_UNCONFIRMED_X 400 error will be returned, where X is the length of the verification code that was just sent to the email.
Use account.confirmPasswordEmail to enter the received verification code and enable the recovery email.
Use account.resendPasswordEmail to resend the verification code.
Use account.cancelPasswordEmail to cancel the verification code.

To get the current recovery email, use account.getPasswordSettings.

Email recovery

In order to recover a forgotten 2FA password, an email must be sent to the previously specified address using the auth.requestPasswordRecovery method.
Then use auth.recoverPassword with the received code to delete the current 2FA password, to set a new one follow these instructions.

Error handling

There will be errors when working with the API, and they must be correctly handled on the client.

An error is characterized by several parameters:

Error Code

Numerical value similar to HTTP status. Contains information on the type of error that occurred: for example, a data input error, privacy error, or server error. This is a required parameter.

Error Type

A string literal in the form of `/[A-Z_0-9]+/`, which summarizes the problem. For example, `AUTH_KEY_UNREGISTERED`. This is an optional parameter.

Error Constructors

There should be a way to handle errors that are returned in rpc_error constructors.

Below is a list of error codes and their meanings:

303 SEE_OTHER

The request must be repeated, but directed to a different data center.

Examples of Errors:

- FILE_MIGRATE_X: the file to be accessed is currently stored in a different data center.
- PHONE_MIGRATE_X: the phone number a user is trying to use for authorization is associated with a different data center.
- NETWORK_MIGRATE_X: the source IP address is associated with a different data center (for registration)
- USER_MIGRATE_X: the user whose identity is being used to execute queries is associated with a different data center (for registration)

In all these cases, the error description's string literal contains the number of the data center (instead of the X) to which the repeated query must be sent.
More information about redirects between data centers »

400 BAD_REQUEST

The query contains errors. In the event that a request was created using a form and contains user generated data, the user should be notified that the data must be corrected before the query is repeated.

Examples of Errors:

- FIRSTNAME_INVALID: The first name is invalid
- LASTNAME_INVALID: The last name is invalid
- PHONE_NUMBER_INVALID: The phone number is invalid
- PHONE_CODE_HASH_EMPTY: phone_code_hash is missing
- PHONE_CODE_EMPTY: phone_code is missing
- PHONE_CODE_EXPIRED: The confirmation code has expired
- API_ID_INVALID: The api_id/api_hash combination is invalid
- PHONE_NUMBER_OCCUPIED: The phone number is already in use
- PHONE_NUMBER_UNOCCUPIED: The phone number is not yet being used
- USERS_TOO_FEW: Not enough users (to create a chat, for example)
- USERS_TOO_MUCH: The maximum number of users has been exceeded (to create a chat, for example)
- TYPE_CONSTRUCTOR_INVALID: The type constructor is invalid
- FILE_PART_INVALID: The file part number is invalid
- FILE_PARTS_INVALID: The number of file parts is invalid
- FILE_PART_X_MISSING: Part X (where X is a number) of the file is missing from storage
- MD5_CHECKSUM_INVALID: The MD5 checksums do not match
- PHOTO_INVALID_DIMENSIONS: The photo dimensions are invalid
- FIELD_NAME_INVALID: The field with the name FIELD_NAME is invalid
- FIELD_NAME_EMPTY: The field with the name FIELD_NAME is missing
- MSG_WAIT_FAILED: A waiting call returned an error

401 UNAUTHORIZED

There was an unauthorized attempt to use functionality available only to authorized users.

Examples of Errors:

- AUTH_KEY_UNREGISTERED: The key is not registered in the system
- AUTH_KEY_INVALID: The key is invalid
- USER_DEACTIVATED: The user has been deleted/deactivated
- SESSION_REVOKED: The authorization has been invalidated, because of the user terminating all sessions
- SESSION_EXPIRED: The authorization has expired
- AUTH_KEY_PERM_EMPTY: The method is unavailble for temporary authorization key, not bound to permanent

403 FORBIDDEN

Privacy violation. For example, an attempt to write a message to someone who has blacklisted the current user.

404 NOT_FOUND

An attempt to invoke a non-existent object, such as a method.

406 NOT_ACCEPTABLE

Similar to 400 BAD_REQUEST, but the app should not display any error messages to user in UI as a result of this response. The error message will be delivered via updateServiceNotification instead.

420 FLOOD

The maximum allowed number of attempts to invoke the given method with the given input parameters has been exceeded. For example, in an attempt to request a large number of text messages (SMS) for the same phone number.

Error Example:

- FLOOD_WAIT_X: A wait of X seconds is required (where X is a number)

500 INTERNAL

An internal server error occurred while a request was being processed; for example, there was a disruption while accessing a database or file storage.

If a client receives a 500 error, or you believe this error should not have occurred, please collect as much information as possible about the query and error and send it to the developers.

Other Error Codes

If a server returns an error with a code other than the ones listed above, it may be considered the same as a 500 error and treated as an internal server error.

Working with Different Data Centers

The servers are divided into several data centers (hereinafter "DCs") in different parts of the world.
A complete list of proxy access points for these DCs may be obtained using help.getConfig:

```
dcOption#2ec2a43c id:int hostname:string ip_address:string port:int = DcOption;
config#232d5905 date:int test_mode:Bool this_dc:int dc_options:Vector<DcOption> chat_size_max:int = Config;
---functions---
help.getConfig#c4f9186b = Config;
```

In this context, **this_dc** is the number of the current DC, **dc_options** is a list of all DCs available at the moment, each of which has an **id**, **ip**, and **port** for establishing a connection. Please note that **ip** and **port** may change frequently, based on proxy server load and the user's current location.

To optimize client communication with the API, each client must use the connection to the closest access point for its main queries (sending messages, getting contacts, etc.). Therefore, knowing how to select a DC is required before communicating with the API.

Registration/Authorization

The auth.sendCode method is the basic entry point when registering a new user or authorizing an existing user. 95% of all redirection cases to a different DC will occure when invoking this method.

The client does not yet know which DC it will be associated with; therefore, it establishes an encrypted connection to a random address and sends its query to that address.
Having received a **phone_number** from a client, we can find out whether or not it is registered in the system. If it is, then, if necessary, instead of sending a text message, we request that it establish a connection with a different DC first (PHONE_MIGRATE_X error).
If we do not yet have a user with this number, we examine its IP-address. We can use it to identify the closest DC. Again, if necessary, we redirect the user to a different DC (NETWORK_MIGRATE_X error).

Testing Redirects

There are reserved phone number prefixes to test the correctness of the application's handling of redirects between DCs. If you wish to emulate an application of a user associated with DC number X, it is sufficient to specify the phone number as `99966XYYYY`, where YYYY are random numbers, when registering the user. A user like this would always get XXXXX as the confirmation code (the DC number, repeated five times).
Do not store any important or private information in such test users' messages; anyone can make use of the simplified authorization mechanism.

File Access

A file saved by a user with upload.saveFilePart will be available for direct download only from the DC where the query was executed. That is why each file has a **dc_id** parameter:

```
fileLocation#53d69076 dc_id:int volume_id:long local_id:int secret:long = FileLocation;
```

To download the file, an encrypted connection to DC **dc_id** must be established and used to execute the upload.getFile query.
If an attempt is made to download the file over a wrong connection, the FILE_MIGRATE_X error will be returned.

Please note that encryption keys are not copied between DCs; therefore, the process of establishing an encrypted connection is started from the very beginning for each new DC. An issued auth_key can be associated with the current authorized user by using an authorization transfer.

User Migration

During the process of working with the API, user information is accumulated in the DC with which the user is associated. This is the reason a user cannot be associated with a different DC by means of the client. However, in the future, during prolonged communication from an unusual location, we may decide that the user's data must be moved to a different DC. After some time, the data will be copied and the association will be updated. Once this happens, when executing any query transmitted to the old DC, the API will return the USER_MIGRATE_X error. The client will then have to establish a connection with the new DC and repeat the query.

Authorization Transfer

The following methods can be used to eliminate the need for users to enter the code from a text message every time:

```
auth.exportedAuthorization#df969c2d id:int bytes:bytes = auth.ExportedAuthorization;
auth.authorization#f6b673a4 expires:int user:User = auth.Authorization;
---functions---
auth.importAuthorization#e3ef9613 id:int bytes:bytes = auth.Authorization;
auth.exportAuthorization#e5bfffcd dc_id:int = auth.ExportedAuthorization;
```

auth.exportAuthorization must be executed in the current DC (the DC with which a connection has already been established), passing in **dc_id** as the value for the new DC. The method should return the user identifier and a long string of random data. An import operation can be performed at the new DC by sending it what was received. Queries requiring authorization can then be successfully executed in the new DC.

Working with Updates

When a client is being actively used, events will occur that affect the current user and that they must learn about as soon as possible, e.g. when a new message is received. To eliminate the need for the client itself to periodically download these events, there is an update delivery mechanism in which the server sends the user notifications over one of its available connections with the client.

Subscribing to Updates

Update events are sent to an authorized user into the last active connection (except for connections needed for downloading / uploading files).

So to start receiving updates the client needs to init connection and call API method, e.g. to fetch current state.

Event sequences

All events are received from the socket as a sequence of TL-serialized Updates objects, which might be optionally gzip-compressed in the same way as responses to queries.

Each Updates object may contain single or multiple Update objects, representing different events happening.

In order to apply all updates in precise order and to guarantee that no update is missed or applied twice there is `seq` attribute in Updates constructors, and `pts` (with `pts_count`) or `qts` attributes in Update constructors. The client must use those attributes values in combination with locally stored state to correctly apply incoming updates.

When a gap in updates sequence occurs, it must be filled via calling one of the API methods. More below »

Updates sequence

As said earlier, each payload with updates has a TL-type Updates. It can be seen from the schema below that this type has several constructors.

```
updatesTooLong#e317af7e = Updates;
updateShort#78d4dec1 update:Update date:int = Updates;
updateShortMessage#914fbf11 flags:# out:flags.1?true mentioned:flags.4?true media_unread:flags.5?true silent:fl
updateShortChatMessage#16812688 flags:# out:flags.1?true mentioned:flags.4?true media_unread:flags.5?true silen
updateShortSentMessage#11f1331c flags:# out:flags.1?true id:int pts:int pts_count:int date:int media:flags.9?Me
updatesCombined#725b04c3 updates:Vector<Update> users:Vector<User> chats:Vector<Chat> date:int seq_start:int se
updates#74ae4240 updates:Vector<Update> users:Vector<User> chats:Vector<Chat> date:int seq:int = Updates;
```

updatesTooLong indicates that there are too many events pending to be pushed to the client, so one needs to fetch them manually.

Events inside updateShort constructors, normally, have lower priority and are broadcast to a large number of users, i.e. one of the chat participants started entering text in a big conversation (updateChatUserTyping).

The updateShortMessage, updateShortSentMessage and updateShortChatMessage constructors are redundant but help significantly reduce the transmitted message size for 90% of the updates. They should be transformed to updateShort upon receiving.

Two remaining constructors updates and updatesCombined are part of the Updates sequence. Both of them have `seq` attribute, which indicates the remote Updates state after the generation of the Updates, and `seq_start` indicates the remote Updates state after the *first* of the Updates in the packet is generated. For updates, `seq_start` attribute is omitted, because it is assumed that it is always equal to `seq`.

Message-related event sequences

Each *event* related to a message box (message created, message edited, message deleted, etc) is identified by a unique autoincremented *pts* (or *qts* in case of secret chats).

Each message box can be considered as some server-side DB table that stores messages and events associated with them.
All boxes are completely independent, and each pts sequence is tied to just one box (see below).

Update object may contain info about *multiple events* (for example, updateDeleteMessages).
That's why all single updates might have *pts_count* parameter indicating the *number of events* contained in the received *update* (with some exceptions, in this case, the *pts_count* is considered to be `0`).

Each channel and supergroup has its message box and *its event sequence* as a result; private chats and legacy groups of one user have another *common event sequence*.
User's Secret chats have yet another *common secret event sequence*.

To recap, the client has to take care of the integrity of the following sequences to properly handle updates:

- Updates sequence (seq)
  - Common message box sequence (pts)
  - Secret message box sequence (qts)
  - Channel message box sequence 1 (pts)
  - Channel message box sequence 2 (pts)
  - Channel message box sequence 3 (pts)
  - and so on…

Fetching state

The *common* update state is represented by the updates.State constructor.
When the user logs in for the first time, call to updates.getState has to be made to store the latest update state (which will not be the absolute initial state, just the latest state at the current time).
The common update state can also be fetched from updates.differenceTooLong.

The *channel update state* is represented simply by the *pts* of the event sequence: when first logging in, the initial channel state can be obtained from the dialog constructor when fetching dialogs, from the full channel info, or it can be received as an updateChannelTooLong update.

The *secret chat update state* is represented by the *qts* of the secret event sequence, it is contained in the updates.State of the *common update state*.

The *Updates sequence state* is represented by the *date* and *seq* of the *Updates sequence*, it is contained in the updates.State of the *common* update state.

Update handling

Update handling in Telegram clients consists of receiving events, making sure there were no gaps and no events were missed based on the locally stored state of the correspondent event sequence, and then updating the locally stored state based on the parameters received.

When the client receives payload with serialized updates, first of all, it needs to walk through all of the nested Update objects and check if they belong to any of message box sequences (have `pts` or `qts` parameters). Those updates need to be handled separately according to corresponding local state and new `pts` / `qts` values. Details below »

After message box updates are handled, if there are any other updates remaining the client needs to handle them with respect to `seq`. Details below »

`pts`: checking and applying

Here, `local_pts` will be the local state, `pts` will be the remote state, `pts_count` will be the number of events in the

update.
If `local_pts + pts_count === pts`, the update can be applied.
If `local_pts + pts_count > pts`, the update was already applied, and must be ignored.
If `local_pts + pts_count < pts`, there's an update gap that must be filled.

For example, let's assume the client has the following local state for the channel `123456789`:

```
local_pts = 131
```

Now let's assume an updateNewChannelMessage from channel `123456789` is received with `pts = 132` and `pts_count=1`.
Since `local_pts + pts_count === pts`, the total number of events since the last stored state is, in fact, equal to `pts_count`: this means the update can be safely accepted and the remote `pts` applied:

```
local_pts = 132
```

Since:

- `pts` indicates the server state **after** the new channel message events are generated
- `pts_count` indicates the number of events in the new channel update
- The server state **before the new channel message event was generated** has to be: `pts_before = pts - pts_count = 131`, which is, in fact, equal to our local state.

Now let's assume an updateNewChannelMessage from channel `123456789` is received with `pts = 132` and `pts_count=1`.
Since `local_pts + pts_count > pts` (`133 > 132`), the update is skipped because we've already handled this update (in fact, our current `local_pts` was set by this same update, and it was resent twice due to network issues or other issues).

Now let's assume an updateDeleteChannelMessages from channel `123456789` is received with `pts = 140` and `pts_count=5`.
Since `local_pts + pts_count < pts` (`137 < 140`), this means that updates were missed, and the gap must be recovered.

Secret chats

The whole process is very similar for secret chats, but there is `qts` instead of `pts`, and events are never grouped, so it's assumed that `qts_count` is always equal to 1.

`seq`: checking and applying

On top level when handling received updates and updatesCombined there are three possible cases:
If `local_seq + 1 === seq_start`, the updates can be applied.
If `local_seq + 1 > seq_start`, the updates were already applied, and must be ignored.
If `local_seq + 1 < seq_start`, there's an updates gap that must be filled (updates.getDifference must be used as with common and secret event sequences).

If the updates were applied, local *Updates state* must be updated with `seq` and `date` from the constructor.

For all the other Updates type constructors there is no need to check `seq` or change a local state.

Recovering gaps

To do this, updates.getDifference (common/secret state) or updates.getChannelDifference (channel state) with the respective local states must be called.
These methods should also be called on startup, to fetch new updates (preferably with some flags to reduce server load, see the method's docs).
Manually obtaining updates is also required in the following situations:

- Loss of sync: a gap was found in **seq** / **pts** / **qts** (as described above). It may be useful to wait up to 0.5 seconds in this situation and abort the sync in case a new update arrives, that fills the gap.
- Session loss on the server: the client receives a new session created notification. This can be caused by garbage collection on the MTProto server or a server reboot.
- Incorrect update: the client cannot deserialize the received data.
- Incomplete update: the client is missing data about a chat/user from one of the shortened constructors, such as updateShortChatMessage, etc.
- Long period without updates: no updates for 15 minutes or longer.
- The server requests the client to fetch the difference using updateChannelTooLong or updatesTooLong.

When calling updates.getDifference if the updates.differenceSlice constructor is returned in response, the full difference was too large to be received in one request. The intermediate status, **intermediate_state**, must be saved on the client and the query must be repeated, using the intermediate status as the current status.

To fetch the updates difference of a channel, updates.getChannelDifference is used.
If the difference is too large to be received in one request, the `final` flag of the result is **not** set (see docs).
The intermediate status, represented by the **pts**, must be saved on the client and the query must be repeated, using the intermediate status as the current status.

For perfomance reasons and for better user experience, client can set maximum gap size to be filled: `pts_total_limit` parameter of updates.getDifference and `limit` parameter for updates.getChannelDifference can be used.

If the gap is too large and there are too many updates to fetch, a `*TooLong` constructor will be returned. In this case, the client must re-fetch the state, re-start fetching updates from that state and follow the instructions that can be found here.

It is recommended to use limit `10-100` for channels and `1000-10000` otherwise.

Example implementations

Implementations also have to take care to postpone updates received via the socket while filling gaps in the event and Update sequences, as well as avoid filling gaps in the same sequence.

Example implementations: tdlib, MadelineProto.

An interesting and easy way this can be implemented, instead of using various locks, is by running background loops, like in MadelineProto ».

PUSH Notifications about Updates

If a client does not have an active connection at the time of an event, PUSH Notifications will also be useful.

Handling PUSH-notifications

Configuring the application

To be able to send APNS notifications to Apple servers or GCM notifications to Google servers, application certificates (APNS) or an application key (GCM) must be specified in the application settings.

Subscribing to notifications

To subscribe to notifications, the client must invoke the account.registerDevice query, passing in **token_type** and **token** as parameters that identify the current device. It is useful to repeat this query at least once every 24 hours or when restarting the application. Use account.unregisterDevice to unsubscribe.

The following modes are supported:

- `1` – APNS (device token for apple push)
- `2` – FCM (firebase token for google firebase)
- `3` – MPNS (channel URI for microsoft push)
- `4` – Deprecated: Simple push (endpoint for firefox's simple push API)
- `5` – Ubuntu phone (token for ubuntu push)
- `6` – Blackberry (token for blackberry push)
- `7` – MTProto separate session
- `8` – WNS (windows push)
- `9` – APNS VoIP (token for apple push VoIP)
- `10` – Web push (web push, see below)
- `11` – MPNS VoIP (token for microsoft push VoIP)
- `12` – Tizen (token for tizen push)

For `10` web push, the token must be a JSON-encoded object with the following keys:

- `endpoint` : Absolute URL exposed by the push service where the application server can send push messages
- `keys` : P-256 elliptic curve Diffie-Hellman parameters in the following object
  - `p256dh` : Base64url-encoded P-256 elliptic curve Diffie-Hellman public key
  - `auth` : Base64url-encoded authentication secret

Notification encryption

For FCM and APNS VoIP, an optional encryption key used to encrypt push notifications can be passed to account.registerDevice (`secret`). This key (`auth_key`) is used to encrypt the payloads using MTProto v2.

The FCM payload will be a JSON payload, containing a `p` field with the base64-encoded encrypted MTProto payload. After decryption, the result will be a JSON object, prefixed by a 32-bit little-endian integer with the length of the JSON payload. As usual, make sure to follow all security checks as described in the MTProto docs.

Example implementation.

As mentioned above, payloads can also be encrypted using P-256 Elliptic Curve Diffie-Hellman when using web push.

Notification structure

An (optionally encrypted) notification is provided as a JSON object in the following format:

```
{
  "data": {
    "loc_key": "CHAT_MESSAGE_CONTACT",
    "loc_args": ["John Doe", "My magical group", "Contact Exchange"],
    "user_id": 14124122,
    "custom": {
      "chat_id": 241233,
      "msg_id": 123
    },
    "sound": "sound1.mp3",
  }
}
```

Each notification has several parameters that describe it.

Notification type

data.loc_key – A string literal in the form `/[A-Z_0-9]+/` , which summarizes the notification. For example, `CHAT_MESSAGE_TEXT` .

Notification text arguments

data.loc_args – A list or arguments which, when inserted into a template, produce a readable notification.

Custom parameters

data.custom – Parameters which are be passed into the application when a notification is opened.

Sound

data.sound – The name of an audio file to be played.

User id

data.user_id – ID of the account to which the PUSH notification should be delivered, in case of clients with multiple accounts active and running.

Processing notifications

In principle, **data.loc_key**, **data.custom**, and an Internet connection are sufficient to generate a notification. Obviously, if possible, when generating a visual notification you need not use all of the transmitted data and may rely on the information already stored on the client. But if a user or a chat is not cached locally, the values passed in loc_args may also be used. **data.user_id** is the ID of the account to which the PUSH notification should be delivered, in case of clients with multiple accounts active and running.

Service notifications

The following notifications can be used to update app settings.

Type	Extra custom arguments	Description
DC_UPDATE	**dc** – number of the data-center **addr** – server address with port number in the format `111.112.113.114:443`	In case the client gets this notification, it is necessary to add the received server address to the list of possible addresses. In case the address of the first DC was passed (`dc=1`), it is recommended to call it immediately using help.getConfig to update dc-configuration.

MESSAGE_DELETED	**channel_id**: For channels and supergroups, Channel/supergroup identifier **chat_id**: For chats, Chat identifier **from_id**: For PMs, Author identifier **messages**: Comma-separated IDs of messages that were deleted	Messages were deleted, remove multiple notifications for this chat
READ_HISTORY	**channel_id**: For channels and supergroups, Channel/supergroup identifier **chat_id**: For chats, Chat identifier **from_id**: For PMs, Author identifier **max_id**: Maximum ID of read messages	Message history was read, remove multiple notifications for this chat
GEO_LIVE_PENDING		Any of the live locations currently being shared should be updated
SESSION_REVOKE		Logout and remove DB for specified session by **data.user_id**, only apply if coming from an MTProto-encrypted payload
MESSAGE_MUTED		Sent rarely, every 10th message in chats or once per 15 sec in PM, to update badge or download messages

Possible Notifications

Type	Template example	Arguments	Extra custom arguments
MESSAGE_TEXT	{1}: {2}	1. Message author 2. Message body	**attachb64**: Base64-encoded version of the attached media **chat_from_id**: Groups only, message author identifier **edit_date**: When was the message last edited **from_id**: Author identifier **mention**: Whether the user was mentioned in the message **msg_id**: ID of the message **silent**: Whether the message was posted silently (no notification should be issued)
MESSAGE_NOTEXT	{1} sent you a message	1. Message author	**attachb64**: Base64-encoded version of the attached media **chat_from_id**: Groups only, message author identifier **edit_date**: When was the message last edited **from_id**: Author identifier **mention**: Whether the user was mentioned in the message **msg_id**: ID of the message **silent**: Whether the message was posted silently (no notification should be issued)
MESSAGE_PHOTO	{1} sent you a photo	1. Message author	**attachb64**: Base64-encoded version of the attached media **chat_from_id**: Groups only, message author identifier **edit_date**: When was the message last edited **from_id**: Author identifier **mention**: Whether the user was mentioned in the message **msg_id**: ID of the message **silent**: Whether the message was posted silently (no notification should be issued)
MESSAGE_PHOTO_SECRET	{1} sent you a self-destructing photo	1. User name	**attachb64**: Base64-encoded version of the attached media **chat_from_id**: Groups only, message author identifier **edit_date**: When was the message last edited **from_id**: Author identifier **mention**: Whether the user was mentioned in the message **msg_id**: ID of the message **silent**: Whether the message was posted silently (no notification should be issued)
MESSAGE_VIDEO	{1} sent you a video	1. Message author	**attachb64**: Base64-encoded version of the attached media **chat_from_id**: Groups only, message author identifier **edit_date**: When was the message last edited **from_id**: Author identifier **mention**: Whether the user was mentioned in the message **msg_id**: ID of the message **silent**: Whether the message was posted silently (no notification should be issued)
MESSAGE_VIDEO_SECRET	{1} sent you a self-destructing video	1. User name	**attachb64**: Base64-encoded version of the attached media **chat_from_id**: Groups only, message author identifier **edit_date**: When was the message last edited **from_id**: Author identifier **mention**: Whether the user was mentioned in the message **msg_id**: ID of the message **silent**: Whether the message was posted silently (no notification should be issued)
MESSAGE_SCREENSHOT	{1} took a screenshot	1. User name	**attachb64**: Base64-encoded version of the attached media **chat_from_id**: Groups only, message author identifier **edit_date**: When was the message last edited **from_id**: Author identifier **mention**: Whether the user was mentioned in the message

			msg_id: ID of the message **silent**: Whether the message was posted silently (no notification should be issued)
MESSAGE_ROUND	{1} sent you a video message	1. User name	**attachb64**: Base64-encoded version of the attached media **chat_from_id**: Groups only, message author identifier **edit_date**: When was the message last edited **from_id**: Author identifier **mention**: Whether the user was mentioned in the message **msg_id**: ID of the message **silent**: Whether the message was posted silently (no notification should be issued)
MESSAGE_DOC	{1} sent you a file	1. User name	**attachb64**: Base64-encoded version of the attached media **chat_from_id**: Groups only, message author identifier **edit_date**: When was the message last edited **from_id**: Author identifier **mention**: Whether the user was mentioned in the message **msg_id**: ID of the message **silent**: Whether the message was posted silently (no notification should be issued)
MESSAGE_STICKER	{1} sent you a {2}sticker	1. User name 2. Sticker type	**attachb64**: Base64-encoded version of the attached media **chat_from_id**: Groups only, message author identifier **edit_date**: When was the message last edited **from_id**: Author identifier **mention**: Whether the user was mentioned in the message **msg_id**: ID of the message **silent**: Whether the message was posted silently (no notification should be issued)
MESSAGE_AUDIO	{1} sent you a voice message	1. Message author	**attachb64**: Base64-encoded version of the attached media **chat_from_id**: Groups only, message author identifier **edit_date**: When was the message last edited **from_id**: Author identifier **mention**: Whether the user was mentioned in the message **msg_id**: ID of the message **silent**: Whether the message was posted silently (no notification should be issued)
MESSAGE_CONTACT	{1} shared a contact {2} with you	1. User name 2. Contact name	**attachb64**: Base64-encoded version of the attached media **chat_from_id**: Groups only, message author identifier **edit_date**: When was the message last edited **from_id**: Author identifier **mention**: Whether the user was mentioned in the message **msg_id**: ID of the message **silent**: Whether the message was posted silently (no notification should be issued)
MESSAGE_GEO	{1} sent you a map	1. Message author	**attachb64**: Base64-encoded version of the attached media **chat_from_id**: Groups only, message author identifier **edit_date**: When was the message last edited **from_id**: Author identifier **mention**: Whether the user was mentioned in the message **msg_id**: ID of the message **silent**: Whether the message was posted silently (no notification should be issued)
MESSAGE_GEOLIVE	{1} started sharing their live location	1. User name	**attachb64**: Base64-encoded version of the attached media **chat_from_id**: Groups only, message author identifier **edit_date**: When was the message last edited **from_id**: Author identifier **mention**: Whether the user was mentioned in the message **msg_id**: ID of the message **silent**: Whether the message was posted silently (no notification should be issued)
MESSAGE_POLL	{1} sent you a poll {2}	1. User name 2. Poll name	**attachb64**: Base64-encoded version of the attached media **chat_from_id**: Groups only, message author identifier **edit_date**: When was the message last edited **from_id**: Author identifier **mention**: Whether the user was mentioned in the message **msg_id**: ID of the message **silent**: Whether the message was posted silently (no notification should be issued)
MESSAGE_GIF	{1} sent you a GIF	1. User name	**attachb64**: Base64-encoded version of the attached media **chat_from_id**: Groups only, message author identifier **edit_date**: When was the message last edited **from_id**: Author identifier **mention**: Whether the user was mentioned in the message **msg_id**: ID of the message **silent**: Whether the message was posted silently (no notification should be issued)
MESSAGE_GAME	{1} invited you to play {2}	1. User name 2. Game name	**attachb64**: Base64-encoded version of the attached media **chat_from_id**: Groups only, message author identifier **edit_date**: When was the message last edited **from_id**: Author identifier **mention**: Whether the user was mentioned in the message

			msg_id: ID of the message **silent**: Whether the message was posted silently (no notification should be issued)
MESSAGE_GAME_SCORE	{1} scored {3} in game {2}	1. User name 2. Game name 3. Score	**attach64**: Base64-encoded version of the attached media **chat_from_id**: Groups only, message author identifier **edit_date**: When was the message last edited **from_id**: Author identifier **mention**: Whether the user was mentioned in the message **msg_id**: ID of the message **silent**: Whether the message was posted silently (no notification should be issued)
MESSAGE_INVOICE	{1} sent you an invoice for {2}	1. User name 2. Product	**attachb64**: Base64-encoded version of the attached media **chat_from_id**: Groups only, message author identifier **edit_date**: When was the message last edited **from_id**: Author identifier **mention**: Whether the user was mentioned in the message **msg_id**: ID of the message **silent**: Whether the message was posted silently (no notification should be issued)
MESSAGE_FWDS	{1} forwarded you {2} messages	1. User name 2. Number of messages that were forwarded	**attachb64**: Base64-encoded version of the attached media (related to the first message) **chat_from_id**: Groups only, message author identifier (related to the first message) **edit_date**: When was the message last edited (related to the first message) **from_id**: Author identifier (related to the first message) **mention**: Whether the user was mentioned in the message (related to the first message) **msg_id**: ID of the message (related to the first message) **silent**: Whether the message was posted silently (no notification should be issued) (related to the first message)
MESSAGE_PHOTOS	{1} sent you {2} photos	1. User name 2. Number of photos that were sent	**attachb64**: Base64-encoded version of the attached media (related to the first message) **chat_from_id**: Groups only, message author identifier (related to the first message) **edit_date**: When was the message last edited (related to the first message) **from_id**: Author identifier (related to the first message) **mention**: Whether the user was mentioned in the message (related to the first message) **msg_id**: ID of the message (related to the first message) **silent**: Whether the message was posted silently (no notification should be issued) (related to the first message)
MESSAGE_VIDEOS	{1} sent you {2} videos	1. User name 2. Number of videos that were sent	**attachb64**: Base64-encoded version of the attached media (related to the first message) **chat_from_id**: Groups only, message author identifier (related to the first message) **edit_date**: When was the message last edited (related to the first message) **from_id**: Author identifier (related to the first message) **mention**: Whether the user was mentioned in the message (related to the first message) **msg_id**: ID of the message (related to the first message) **silent**: Whether the message was posted silently (no notification should be issued) (related to the first message)
MESSAGES	{1} sent you an album	1. User name	**from_id**: author identifier
CHANNEL_MESSAGE_TEXT	{1}: {2}	1. Channel name 2. Message body	**attachb64**: Base64-encoded version of the attached media **channel_id**: Channel/supergroup identifier **chat_from_id**: Groups only, message author identifier **edit_date**: When was the message last edited **mention**: Whether the user was mentioned in the message **msg_id**: ID of the message **silent**: Whether the message was posted silently (no notification should be issued)
CHANNEL_MESSAGE_NOTEXT	{1} posted a message	1. Channel name	**attachb64**: Base64-encoded version of the attached media **channel_id**: Channel/supergroup identifier **chat_from_id**: Groups only, message author identifier **edit_date**: When was the message last edited **mention**: Whether the user was mentioned in the message **msg_id**: ID of the message **silent**: Whether the message was posted silently (no notification should be issued)
CHANNEL_MESSAGE_PHOTO	{1} posted a photo	1. Channel name	**attachb64**: Base64-encoded version of the attached media **channel_id**: Channel/supergroup identifier **chat_from_id**: Groups only, message author identifier **edit_date**: When was the message last edited **mention**: Whether the user was mentioned in the message **msg_id**: ID of the message **silent**: Whether the message was posted silently (no notification should be issued)
CHANNEL_MESSAGE_VIDEO	{1} posted a video	1. Channel name	**attachb64**: Base64-encoded version of the attached media **channel_id**: Channel/supergroup identifier **chat_from_id**: Groups only, message author identifier

			edit_date: When was the message last edited **mention**: Whether the user was mentioned in the message **msg_id**: ID of the message **silent**: Whether the message was posted silently (no notification should be issued)
CHANNEL_MESSAGE_ROUND	{1} posted a video message	1. Channel name	**attachb64**: Base64-encoded version of the attached media **channel_id**: Channel/supergroup identifier **chat_from_id**: Groups only, message author identifier **edit_date**: When was the message last edited **mention**: Whether the user was mentioned in the message **msg_id**: ID of the message **silent**: Whether the message was posted silently (no notification should be issued)
CHANNEL_MESSAGE_DOC	{1} posted a file	1. Message author	**attachb64**: Base64-encoded version of the attached media **channel_id**: Channel/supergroup identifier **chat_from_id**: Groups only, message author identifier **edit_date**: When was the message last edited **mention**: Whether the user was mentioned in the message **msg_id**: ID of the message **silent**: Whether the message was posted silently (no notification should be issued)
CHANNEL_MESSAGE_STICKER	{1} posted a {2}sticker	1. Channel name 2. Channel name	**attachb64**: Base64-encoded version of the attached media **channel_id**: Channel/supergroup identifier **chat_from_id**: Groups only, message author identifier **edit_date**: When was the message last edited **mention**: Whether the user was mentioned in the message **msg_id**: ID of the message **silent**: Whether the message was posted silently (no notification should be issued)
CHANNEL_MESSAGE_AUDIO	{1} posted a voice message	1. Message author	**attachb64**: Base64-encoded version of the attached media **channel_id**: Channel/supergroup identifier **chat_from_id**: Groups only, message author identifier **edit_date**: When was the message last edited **mention**: Whether the user was mentioned in the message **msg_id**: ID of the message **silent**: Whether the message was posted silently (no notification should be issued)
CHANNEL_MESSAGE_CONTACT	{1} posted a contact {2}	1. Message author 2. Contact name	**attachb64**: Base64-encoded version of the attached media **channel_id**: Channel/supergroup identifier **chat_from_id**: Groups only, message author identifier **edit_date**: When was the message last edited **mention**: Whether the user was mentioned in the message **msg_id**: ID of the message **silent**: Whether the message was posted silently (no notification should be issued)
CHANNEL_MESSAGE_GEO	{1} posted a map	1. Channel name	**attachb64**: Base64-encoded version of the attached media **channel_id**: Channel/supergroup identifier **chat_from_id**: Groups only, message author identifier **edit_date**: When was the message last edited **mention**: Whether the user was mentioned in the message **msg_id**: ID of the message **silent**: Whether the message was posted silently (no notification should be issued)
CHANNEL_MESSAGE_GEOLIVE	{1} posted a live location	1. Channel name	**attachb64**: Base64-encoded version of the attached media **channel_id**: Channel/supergroup identifier **chat_from_id**: Groups only, message author identifier **edit_date**: When was the message last edited **mention**: Whether the user was mentioned in the message **msg_id**: ID of the message **silent**: Whether the message was posted silently (no notification should be issued)
CHANNEL_MESSAGE_POLL	{1} posted a poll {2}	1. Channel name 2. Channel name	**attachb64**: Base64-encoded version of the attached media **channel_id**: Channel/supergroup identifier **chat_from_id**: Groups only, message author identifier **edit_date**: When was the message last edited **mention**: Whether the user was mentioned in the message **msg_id**: ID of the message **silent**: Whether the message was posted silently (no notification should be issued)
CHANNEL_MESSAGE_GIF	{1} posted a GIF	1. Channel name	**attachb64**: Base64-encoded version of the attached media **channel_id**: Channel/supergroup identifier **chat_from_id**: Groups only, message author identifier **edit_date**: When was the message last edited **mention**: Whether the user was mentioned in the message **msg_id**: ID of the message **silent**: Whether the message was posted silently (no notification should be issued)
CHANNEL_MESSAGE_GAME	{1} invited you to play {2}	1. Message author 2. Game name	**attachb64**: Base64-encoded version of the attached media **channel_id**: Channel/supergroup identifier **chat_from_id**: Groups only, message author identifier

			edit_date: When was the message last edited **mention**: Whether the user was mentioned in the message **msg_id**: ID of the message **silent**: Whether the message was posted silently (no notification should be issued)
CHANNEL_MESSAGE_GAME_SCORE	{1} scored {3} in game {2}	1. User 2. Game name 3. Score	**attachb64**: Base64-encoded version of the attached media **channel_id**: Channel/supergroup identifier **chat_from_id**: Groups only, message author identifier **edit_date**: When was the message last edited **mention**: Whether the user was mentioned in the message **msg_id**: ID of the message **silent**: Whether the message was posted silently (no notification should be issued)
CHANNEL_MESSAGE_FWDS	{1} posted {2} forwarded messages	1. Message author 2. Number of forwarded messages	**attachb64**: Base64-encoded version of the attached media (related to the first message) **channel_id**: Channel/supergroup identifier (related to the first message) **chat_from_id**: Groups only, message author identifier (related to the first message) **edit_date**: When was the message last edited (related to the first message) **mention**: Whether the user was mentioned in the message (related to the first message) **msg_id**: ID of the message (related to the first message) **silent**: Whether the message was posted silently (no notification should be issued) (related to the first message)
CHANNEL_MESSAGE_PHOTOS	{1} posted {2} photos	1. Channel name 2. Channel name	**attachb64**: Base64-encoded version of the attached media (related to the first message) **channel_id**: Channel/supergroup identifier (related to the first message) **chat_from_id**: Groups only, message author identifier (related to the first message) **edit_date**: When was the message last edited (related to the first message) **mention**: Whether the user was mentioned in the message (related to the first message) **msg_id**: ID of the message (related to the first message) **silent**: Whether the message was posted silently (no notification should be issued) (related to the first message)
CHANNEL_MESSAGE_VIDEOS	{1} posted {2} videos	1. Channel name 2. Number of vidoes that were posted	**attachb64**: Base64-encoded version of the attached media (related to the first message) **channel_id**: Channel/supergroup identifier (related to the first message) **chat_from_id**: Groups only, message author identifier (related to the first message) **edit_date**: When was the message last edited (related to the first message) **mention**: Whether the user was mentioned in the message (related to the first message) **msg_id**: ID of the message (related to the first message) **silent**: Whether the message was posted silently (no notification should be issued) (related to the first message)
CHANNEL_MESSAGES	{1} posted an album	1. Message author	**attachb64**: Base64-encoded version of the attached media (related to the first message) **channel_id**: Channel/supergroup identifier (related to the first message) **chat_from_id**: Groups only, message author identifier (related to the first message) **edit_date**: When was the message last edited (related to the first message) **mention**: Whether the user was mentioned in the message (related to the first message) **msg_id**: ID of the message (related to the first message) **silent**: Whether the message was posted silently (no notification should be issued) (related to the first message)
CHAT_MESSAGE_TEXT	{1}@{2}: {3}	1. Message author 2. Chat name 3. Message body	**attachb64**: Base64-encoded version of the attached media **chat_from_id**: Groups only, message author identifier **chat_id**: Chat identifier **edit_date**: When was the message last edited **mention**: Whether the user was mentioned in the message **msg_id**: ID of the message **silent**: Whether the message was posted silently (no notification should be issued)
CHAT_MESSAGE_NOTEXT	{1} sent a message to the group {2}	1. Message author 2. Chat name	**attachb64**: Base64-encoded version of the attached media **chat_from_id**: Groups only, message author identifier **chat_id**: Chat identifier **edit_date**: When was the message last edited **mention**: Whether the user was mentioned in the message **msg_id**: ID of the message **silent**: Whether the message was posted silently (no notification should be issued)
CHAT_MESSAGE_PHOTO	{1} sent a photo to the group {2}	1. Message author 2. Chat name	**attachb64**: Base64-encoded version of the attached media **chat_from_id**: Groups only, message author identifier **chat_id**: Chat identifier **edit_date**: When was the message last edited **mention**: Whether the user was mentioned in the message **msg_id**: ID of the message **silent**: Whether the message was posted silently (no notification should be issued)
CHAT_MESSAGE_VIDEO	{1} sent a	1. Message	**attachb64**: Base64-encoded version of the

CHAT_MESSAGE_VIDEO	{1} sent a video to the group {2}	1. Message author 2. Chat name	**attachb64**: Base64-encoded version of the attached media **chat_from_id**: Groups only, message author identifier **chat_id**: Chat identifier **edit_date**: When was the message last edited **mention**: Whether the user was mentioned in the message **msg_id**: ID of the message **silent**: Whether the message was posted silently (no notification should be issued)
CHAT_MESSAGE_ROUND	{1} sent a video message to the group {2}	1. User name 2. Group name	**attachb64**: Base64-encoded version of the attached media **chat_from_id**: Groups only, message author identifier **chat_id**: Chat identifier **edit_date**: When was the message last edited **mention**: Whether the user was mentioned in the message **msg_id**: ID of the message **silent**: Whether the message was posted silently (no notification should be issued)
CHAT_MESSAGE_DOC	{1} sent a file to the group {2}	1. User name 2. Group name	**attachb64**: Base64-encoded version of the attached media **chat_from_id**: Groups only, message author identifier **chat_id**: Chat identifier **edit_date**: When was the message last edited **mention**: Whether the user was mentioned in the message **msg_id**: ID of the message **silent**: Whether the message was posted silently (no notification should be issued)
CHAT_MESSAGE_STICKER	{1} sent a {3}sticker to the group {2}	1. User name 2. Group name 3. Sticker type	**attachb64**: Base64-encoded version of the attached media **chat_from_id**: Groups only, message author identifier **chat_id**: Chat identifier **edit_date**: When was the message last edited **mention**: Whether the user was mentioned in the message **msg_id**: ID of the message **silent**: Whether the message was posted silently (no notification should be issued)
CHAT_MESSAGE_AUDIO	{1} sent a voice message to the group {2}	1. Message author 2. Chat name	**attachb64**: Base64-encoded version of the attached media **chat_from_id**: Groups only, message author identifier **chat_id**: Chat identifier **edit_date**: When was the message last edited **mention**: Whether the user was mentioned in the message **msg_id**: ID of the message **silent**: Whether the message was posted silently (no notification should be issued)
CHAT_MESSAGE_CONTACT	{1} shared a contact {3} in the group {2}	1. User name 2. Group name 3. Contact name	**attachb64**: Base64-encoded version of the attached media **chat_from_id**: Groups only, message author identifier **chat_id**: Chat identifier **edit_date**: When was the message last edited **mention**: Whether the user was mentioned in the message **msg_id**: ID of the message **silent**: Whether the message was posted silently (no notification should be issued)
CHAT_MESSAGE_GEO	{1} sent a map to the group {2}	1. Message author 2. Chat name	**attachb64**: Base64-encoded version of the attached media **chat_from_id**: Groups only, message author identifier **chat_id**: Chat identifier **edit_date**: When was the message last edited **mention**: Whether the user was mentioned in the message **msg_id**: ID of the message **silent**: Whether the message was posted silently (no notification should be issued)
CHAT_MESSAGE_GEOLIVE	{1} started sharing their live location with {2}	1. User name 2. Group name	**attachb64**: Base64-encoded version of the attached media **chat_from_id**: Groups only, message author identifier **chat_id**: Chat identifier **edit_date**: When was the message last edited **mention**: Whether the user was mentioned in the message **msg_id**: ID of the message **silent**: Whether the message was posted silently (no notification should be issued)
CHAT_MESSAGE_POLL	{1} sent a poll {3} to the group {2}	1. User name 2. Group name 3. Poll name	**attachb64**: Base64-encoded version of the attached media **chat_from_id**: Groups only, message author identifier **chat_id**: Chat identifier **edit_date**: When was the message last edited **mention**: Whether the user was mentioned in the message **msg_id**: ID of the message **silent**: Whether the message was posted silently (no notification should be issued)
CHAT_MESSAGE_GIF	{1} sent a GIF to the group {2}	1. User name 2. Group name	**attachb64**: Base64-encoded version of the attached media **chat_from_id**: Groups only, message author identifier **chat_id**: Chat identifier **edit_date**: When was the message last edited **mention**: Whether the user was mentioned in the message **msg_id**: ID of the message **silent**: Whether the message was posted silently (no notification should be issued)
CHAT_MESSAGE_GAME	{1} invited	1. User name	**attachb64**: Base64-encoded version of the

CHAT_MESSAGE_GAME	{1} invited the group {2} to play {3}.	1. User name 2. Group name 3. Game name	**attachb64**: Base64-encoded version of the attached media **chat_from_id**: Groups only, message author identifier **chat_id**: Chat identifier **edit_date**: When was the message last edited **mention**: Whether the user was mentioned in the message **msg_id**: ID of the message **silent**: Whether the message was posted silently (no notification should be issued)
CHAT_MESSAGE_GAME_SCORE	{1} scored {4} in game {3} in the group {2}	1. User name 2. Group name 3. Game name 4. Score	**attachb64**: Base64-encoded version of the attached media **chat_from_id**: Groups only, message author identifier **chat_id**: Chat identifier **edit_date**: When was the message last edited **mention**: Whether the user was mentioned in the message **msg_id**: ID of the message **silent**: Whether the message was posted silently (no notification should be issued)
CHAT_MESSAGE_INVOICE	{1} sent an invoice to the group {2} for {3}	1. User name 2. Group name 3. Product name	**attachb64**: Base64-encoded version of the attached media **chat_from_id**: Groups only, message author identifier **chat_id**: Chat identifier **edit_date**: When was the message last edited **mention**: Whether the user was mentioned in the message **msg_id**: ID of the message **silent**: Whether the message was posted silently (no notification should be issued)
CHAT_CREATED	{1} invited you to the group {2}	1. Message author 2. Chat name	**chat_id**: chat identifier
CHAT_TITLE_EDITED	{1} edited the group's {2} name	1. User name 2. Group name	**chat_id**: chat identifier
CHAT_PHOTO_EDITED	{1} edited the group's {2} photo	1. Message author 2. Chat name	**chat_from_id**: Message author identifier **chat_id**: chat identifier **msg_id**: ID of the message
CHAT_ADD_MEMBER	{1} invited {3} to the group {2}	1. Message author 2. Chat name 3. New participant name	**chat_id**: chat identifier
CHAT_ADD_YOU	{1} invited you to the group {2}	1. User name 2. Group name	**chat_id**: chat identifier
CHAT_DELETE_MEMBER	{1} kicked {3} from the group {2}	1. Message author 2. Chat name 3. Dropped participant name	**chat_id**: chat identifier
CHAT_DELETE_YOU	{1} kicked you from the group {2}	1. Message author 2. Chat name	**chat_id**: chat identifier
CHAT_LEFT	{1} has left the group {2}	1. Message author 2. Chat name	**chat_id**: chat identifier
CHAT_RETURNED	{1} has returned to the group {2}	1. Message author 2. Chat name	**chat_id**: chat identifier
CHAT_JOINED	{1} has joined the group {2}	1. User name 2. Group name	**chat_id**: chat identifier
CHAT_MESSAGE_FWDS	{1} forwarded {3} messages to the group {2}	1. User name 2. Group name 3. Number of messages that were forwarded	**attachb64**: Base64-encoded version of the attached media (related to the first message) **chat_from_id**: Groups only, message author identifier (related to the first message) **chat_id**: Chat identifier (related to the first message) **edit_date**: When was the message last edited (related to the first message) **mention**: Whether the user was mentioned in the message (related to the first message) **msg_id**: ID of the message (related to the first message) **silent**: Whether the message was posted silently (no notification should be issued) (related to the first message)
CHAT_MESSAGE_PHOTOS	{1} sent {3} photos to the group {2}	1. User name 2. Group name 3. Number of photos that were sent	**attachb64**: Base64-encoded version of the attached media (related to the first message) **chat_from_id**: Groups only, message author identifier (related to the first message) **chat_id**: Chat identifier (related to the first message) **edit_date**: When was the message last edited (related to the first message) **mention**: Whether the user was mentioned in the message (related to the first message) **msg_id**: ID of the message (related to the first message) **silent**: Whether the message was posted silently (no notification should be issued) (related to the first message)
CHAT_MESSAGE_VIDEOS	{1} sent {3} videos to the group {2}	1. User name 2. Group name 3. Number of videos that were sent	**attachb64**: Base64-encoded version of the attached media (related to the first message) **chat_from_id**: Groups only, message author identifier (related to the first message) **chat_id**: Chat identifier (related to the first message) **edit_date**: When was the message last edited (related to the first message)

			mention: Whether the user was mentioned in the message (related to the first message) **msg_id**: ID of the message (related to the first message) **silent**: Whether the message was posted silently (no notification should be issued) (related to the first message)
CHAT_MESSAGES	{1} sent an album to the group {2}	1. User name 2. Group name	**chat_from_id**: Message author identifier **chat_id**: chat identifier **mention**: Whether the user was mentioned in the message
PINNED_TEXT	{1} pinned " {2}"	1. User name 2. Message body	**attachb64**: Base64-encoded version of the attached media **channel_id**: For channels and supergroups, Channel/supergroup identifier **chat_from_id**: Groups only, message author identifier **chat_id**: For chats, Chat identifier **edit_date**: When was the message last edited **from_id**: For PMs, Author identifier **mention**: Whether the user was mentioned in the message **msg_id**: ID of the message **silent**: Whether the message was posted silently (no notification should be issued)
PINNED_NOTEXT	{1} pinned a message	1. User name	**attachb64**: Base64-encoded version of the attached media **channel_id**: For channels and supergroups, Channel/supergroup identifier **chat_from_id**: Groups only, message author identifier **chat_id**: For chats, Chat identifier **edit_date**: When was the message last edited **from_id**: For PMs, Author identifier **mention**: Whether the user was mentioned in the message **msg_id**: ID of the message **silent**: Whether the message was posted silently (no notification should be issued)
PINNED_PHOTO	{1} pinned a photo	1. User name	**attachb64**: Base64-encoded version of the attached media **channel_id**: For channels and supergroups, Channel/supergroup identifier **chat_from_id**: Groups only, message author identifier **chat_id**: For chats, Chat identifier **edit_date**: When was the message last edited **from_id**: For PMs, Author identifier **mention**: Whether the user was mentioned in the message **msg_id**: ID of the message **silent**: Whether the message was posted silently (no notification should be issued)
PINNED_VIDEO	{1} pinned a video	1. User name	**attachb64**: Base64-encoded version of the attached media **channel_id**: For channels and supergroups, Channel/supergroup identifier **chat_from_id**: Groups only, message author identifier **chat_id**: For chats, Chat identifier **edit_date**: When was the message last edited **from_id**: For PMs, Author identifier **mention**: Whether the user was mentioned in the message **msg_id**: ID of the message **silent**: Whether the message was posted silently (no notification should be issued)
PINNED_ROUND	{1} pinned a video message	1. User name	**attachb64**: Base64-encoded version of the attached media **channel_id**: For channels and supergroups, Channel/supergroup identifier **chat_from_id**: Groups only, message author identifier **chat_id**: For chats, Chat identifier **edit_date**: When was the message last edited **from_id**: For PMs, Author identifier **mention**: Whether the user was mentioned in the message **msg_id**: ID of the message **silent**: Whether the message was posted silently (no notification should be issued)
PINNED_DOC	{1} pinned a file	1. User name	**attachb64**: Base64-encoded version of the attached media **channel_id**: For channels and supergroups, Channel/supergroup identifier **chat_from_id**: Groups only, message author identifier **chat_id**: For chats, Chat identifier **edit_date**: When was the message last edited **from_id**: For PMs, Author identifier **mention**: Whether the user was mentioned in the message **msg_id**: ID of the message **silent**: Whether the message was posted silently (no notification should be issued)
PINNED_STICKER	{1} pinned a {2}sticker	1. User name 2. Sticker type	**attachb64**: Base64-encoded version of the attached media **channel_id**: For channels and supergroups, Channel/supergroup identifier **chat_from_id**: Groups only, message author identifier **chat_id**: For chats, Chat identifier **edit_date**: When was the message last edited **from_id**: For PMs, Author identifier **mention**: Whether the user was mentioned in the message **msg_id**: ID of the message **silent**: Whether the message was posted silently (no notification should be issued)
PINNED_AUDIO	{1} pinned a voice	1. User name	**attachb64**: Base64-encoded version of the attached media

	message		**channel_id**: For channels and supergroups, Channel/supergroup identifier **chat_from_id**: Groups only, message author identifier **chat_id**: For chats, Chat identifier **edit_date**: When was the message last edited **from_id**: For PMs, Author identifier **mention**: Whether the user was mentioned in the message **msg_id**: ID of the message **silent**: Whether the message was posted silently (no notification should be issued)
PINNED_CONTACT	{1} pinned a contact {2}	1. User name 2. User name	**attachb64**: Base64-encoded version of the attached media **channel_id**: For channels and supergroups, Channel/supergroup identifier **chat_from_id**: Groups only, message author identifier **chat_id**: For chats, Chat identifier **edit_date**: When was the message last edited **from_id**: For PMs, Author identifier **mention**: Whether the user was mentioned in the message **msg_id**: ID of the message **silent**: Whether the message was posted silently (no notification should be issued)
PINNED_GEO	{1} pinned a map	1. User name	**attachb64**: Base64-encoded version of the attached media **channel_id**: For channels and supergroups, Channel/supergroup identifier **chat_from_id**: Groups only, message author identifier **chat_id**: For chats, Chat identifier **edit_date**: When was the message last edited **from_id**: For PMs, Author identifier **mention**: Whether the user was mentioned in the message **msg_id**: ID of the message **silent**: Whether the message was posted silently (no notification should be issued)
PINNED_GEOLIVE	{1} pinned a live location	1. User name	**attachb64**: Base64-encoded version of the attached media **channel_id**: For channels and supergroups, Channel/supergroup identifier **chat_from_id**: Groups only, message author identifier **chat_id**: For chats, Chat identifier **edit_date**: When was the message last edited **from_id**: For PMs, Author identifier **mention**: Whether the user was mentioned in the message **msg_id**: ID of the message **silent**: Whether the message was posted silently (no notification should be issued)
PINNED_POLL	{1} pinned a poll {2}	1. User name 2. Poll name	**attachb64**: Base64-encoded version of the attached media **channel_id**: For channels and supergroups, Channel/supergroup identifier **chat_from_id**: Groups only, message author identifier **chat_id**: For chats, Chat identifier **edit_date**: When was the message last edited **from_id**: For PMs, Author identifier **mention**: Whether the user was mentioned in the message **msg_id**: ID of the message **silent**: Whether the message was posted silently (no notification should be issued)
PINNED_GAME	{1} pinned a game	1. User name	**attachb64**: Base64-encoded version of the attached media **channel_id**: For channels and supergroups, Channel/supergroup identifier **chat_from_id**: Groups only, message author identifier **chat_id**: For chats, Chat identifier **edit_date**: When was the message last edited **from_id**: For PMs, Author identifier **mention**: Whether the user was mentioned in the message **msg_id**: ID of the message **silent**: Whether the message was posted silently (no notification should be issued)
PINNED_GAME_SCORE	{1} pinned a game score	1. User name	**attachb64**: Base64-encoded version of the attached media **channel_id**: For channels and supergroups, Channel/supergroup identifier **chat_from_id**: Groups only, message author identifier **chat_id**: For chats, Chat identifier **edit_date**: When was the message last edited **from_id**: For PMs, Author identifier **mention**: Whether the user was mentioned in the message **msg_id**: ID of the message **silent**: Whether the message was posted silently (no notification should be issued)
PINNED_INVOICE	{1} pinned an invoice	1. User name	**attachb64**: Base64-encoded version of the attached media **channel_id**: For channels and supergroups, Channel/supergroup identifier **chat_from_id**: Groups only, message author identifier **chat_id**: For chats, Chat identifier **edit_date**: When was the message last edited **from_id**: For PMs, Author identifier **mention**: Whether the user was mentioned in the message **msg_id**: ID of the message **silent**: Whether the message was posted silently (no notification should be issued)
PINNED_GIF	{1} pinned a	1. User name	**attachb64**: Base64-encoded version of the

	Gif		attached media **channel_id**: For channels and supergroups, Channel/supergroup identifier **chat_from_id**: Groups only, message author identifier **chat_id**: For chats, Chat identifier **edit_date**: When was the message last edited **from_id**: For PMs, Author identifier **mention**: Whether the user was mentioned in the message **msg_id**: ID of the message **silent**: Whether the message was posted silently (no notification should be issued)
CONTACT_JOINED	{1} joined Telegram!	1. Contact name	**contact_id**: contact identifier
AUTH_UNKNOWN	New login from unrecognized device {1}	1. Device name	
AUTH_REGION	New login from unrecognized device {1}, location: {2}	1. Device name 2. Location	
ENCRYPTION_REQUEST	You have a new message		**encryption_id**: secret chat identifier
ENCRYPTION_ACCEPT	You have a new message		**encryption_id**: secret chat identifier
ENCRYPTED_MESSAGE	You have a new message		**encryption_id**: secret chat identifier **random_id**: message identifier
LOCKED_MESSAGE	You have a new message		
PHONE_CALL_REQUEST	{1} is calling you!	1. User name	**call_ah**: Call access hash **call_id**: Call ID
PHONE_CALL_MISSED	You missed a call from {1}	1. User name	
MESSAGE_ANNOUNCEMENT	{1}	1. Announcement	**announcement**: Announcement: roughly equivalent to a message received from the service notifications (Telegram Notifications, id `777000`) user, but must be delivered via push notifications, without contacting the API.

Channels

Channels, chats & supergrups

Channels are a tool for broadcasting your messages to large audiences. They can have an unlimited number of subscribers, they can be public with a permanent URL and each post in a channel has its own view counter. Technically, they are represented by channel constructors.

Supergroups are a powerful tool for building communities and can support up to 200,000 members each. Technically, supergroups are actually channels: they are represented by channel constructors, with the `megagroup` flag set to true.

Channels and supergroup can be created using the channels.createChannel method, by setting the appropriate `broadcast` or `megagroup` flags.
Supergroups can also be assigned a `geo_point` to become geochats.

In previous versions of telegram, only normal groups (represented by chat constructors) could be created using messages.createChat: these groups have fewer features, and can only have 200 members at max.

Migration

To upgrade a legacy group to a supergroup, messages.migrateChat can be used.
The `chats` field of the result will have two objects:

- A chat constructor with a `migrated_to` field, indicating the address of the new supergroup
- The new channel megagroup constructor

When getting full info about the migrated channel, the channelFull object will have `migrated_from_chat_id` and `migrated_from_max_id` fields indicating the original ID of the chat, and the message ID in the original chat at which the group was migrated.

All users of the chat will receive an updateNewMessage from the old chat with a messageService containing a messageActionChatMigrateTo constructor.

All new messages have to be sent to the new supergroup.

When working with migrated groups clients need to handle loading of the message history (as well as search results et cetera) from both the legacy group and the new supergroup. This is done by merging the two messages lists (requested with different Peer values) client side.

Rights

Channels and supergroups allow setting granular permissions both for admins and specific users; channels, supergroups and legacy groups also allow setting global granular permissions for users.

For more info on how to set and modify rights, see here.

Calling API Methods

Layers

Versioning in the API is supported by so-called TL layers.

The need to add a new object constructor or to add/remove a field in a constructor creates a backwards compatibility problem for previous versions of API clients. After all, simply changing a constructor in a schema also changes its number. To address this problem, each schema update is separated into a layer.
A layer is a collection of updated methods or constructors in a TL schema. Each layer is numbered with sequentially increasing numbers starting with 2. The first layer is the base layer — the TL schema without any changes.

There is helper method to let the API know that a client supports Layer `layer`:

```
invokeWithLayer#da9b0d0d {X:Type} layer:int query:!X = X;
```

The helper method **invokeWithLayer** can be used only together with initConnection: the present layer will be saved with all other parameters of the client and future calls will be using this saved value. See more below..

List of Available Layers

Saving Client Info

It is possible to save information about the current client on the server in conjunction with an authorization key. This may help eliminate client-side problems with certain releases on certain devices or with certain localizations, as well as eliminate the need for sending layer information in each call.

The helper method **initConnection** accepts client parameters. This method must be called when first calling the API after the application has restarted or in case the value of one of the parameters could have changed.

initConnection must also be called after each auth.bindTempAuthKey.

When calling this method, the current layer used by the client is also saved (the layer in which initConnection was wrapped is used). After a successful call to initConnection it is no longer necessary to wrap each API call in **invokeWithLayerN**.

Disabling updates

```
invokeWithoutUpdates#bf9459b7 {X:Type} query:!X = X;
```

invokeWithoutUpdates can be used to invoke a request without subscribing the used connection for updates (this is enabled by default for file queries).

Sequential Calls

Sometimes a client needs to transmit several send message method calls to the server all at once in a single message or in several consecutive messages. However, there is a chance that the server may execute these requests out of order (queries are handled by different servers to improve performance, which introduces a degree of randomness to the process).

There are helper methods for making several consecutive API calls without wasting time waiting for a response:

```
invokeAfterMsg#cb9f372d {X:Type} msg_id:long query:!X = X;
invokeAfterMsgs#3dc4b4f0 {X:Type} msg_ids:Vector<long> query:!X = X;
```

They may be used, for example, if a client attempts to send accumulated messages after the Internet connection has been restored after being absent for a long time. In this case, the 32-bit number `0xcb9f372d` must be added before the method number in each call, followed by a 64-bit message identifier, msg_id, which contains the previous call in the queue.
The second method is similar, except it takes several messages that must be waited for.

If the waiting period exceeds 0.5 seconds (this value may change in the future) and no result has appeared, the method will be executed just the same. If any of the queries returns an error, all its dependent queries will also return the 400 MSG_WAIT_FAILED error.

Helper Method Sequence

Important: if the helper methods **invokeAfterMsg** / **invokeAfterMsgs** are used together with **invokeWithLayerN** or other helper methods, **invokeAfterMsg** / **invokeAfterMsgs** must always be the outermost wrapper.

Data Compression

We recommend using gzip compression when making method calls in order to reduce the amount of network traffic.

The schema and constructor information are given in the protocol documentation.

Data Compression when Making a Call

Before transmitting a query, the string containing the entire body of the serialized high-level query (starting with the method number) must be compressed using gzip. If the resulting string is smaller than the original, it makes sense to transmit the gzip_packed constructor.

There is no point in doing the above when transmitting binary multimedia data (photos, videos) or small messages (up to 255 bytes).

Uncompressing Data

By default, the server compresses the response to any call as well as updates, in accordance with the rules stated above. If the gzip_packed constructor is received as a response in rpc_result, then the string that follows must be extracted and uncompressed. Processing then continues on the resulting new string.

Uploading and Downloading Files

When working with the API, it is sometimes necessary to send a relatively large file to the server. For example, when sending a message with a photo/video attachment or when setting the current user's profile picture.

Uploading files

There are a number of API methods to save files. The schema of the types and methods used is presented below:

```
inputFile#f52ff27f id:long parts:int name:string md5_checksum:string = InputFile;
inputFileBig#fa4f0bb5 id:long parts:int name:string md5_checksum:string = InputFile;


inputEncryptedFileUploaded id:long parts:int md5_checksum:string key_fingerprint:int = InputEncryptedFile;
inputEncryptedFileBigUploaded id:long parts:int md5_checksum:string key_fingerprint:int = InputEncryptedFile;

inputSecureFileUploaded#3334b0f0 id:long parts:int md5_checksum:string file_hash:bytes secret:bytes = InputSecu
inputSecureFile#5367e5be id:long access_hash:long = InputSecureFile;

inputMediaUploadedPhoto#1e287d04 flags:# file:InputFile stickers:flags.0?Vector<InputDocument> ttl_seconds:flag
inputMediaUploadedDocument#5b38c6c1 flags:# nosound_video:flags.3?true file:InputFile thumb:flags.2?InputFile m

inputChatUploadedPhoto#94254732 file:InputFile crop:InputPhotoCrop = InputChatPhoto;


---functions---

messages.sendMedia#b8d1262b flags:# silent:flags.5?true background:flags.6?true clear_draft:flags.7?true peer:I
messages.sendEncryptedFile peer:InputEncryptedChat random_id:long data:bytes file:InputEncryptedFile = messages

photos.uploadProfilePhoto#4f32c098 file:InputFile = photos.Photo;

upload.saveFilePart#b304a621 file_id:long file_part:int bytes:bytes = Bool;
upload.saveBigFilePart file_id:long file_part:int file_total_parts:int bytes:bytes = Bool;
```

Before transmitting the contents of the file itself, the file has to be assigned a unique 64-bit client identifier: **file_id**.

The file's binary content is then split into parts. All parts must have the same size (**part_size**) and the following conditions must be met:

- `part_size % 1024 = 0` (divisible by 1KB)
- `524288 % part_size = 0` (512KB must be evenly divisible by **part_size**)

The last part does not have to satisfy these conditions, provided its size is less than **part_size**.

Each part should have a sequence number, **file_part**, with a value ranging from 0 to 2,999.

After the file has been partitioned you need to choose a method for saving it on the server. Use upload.saveBigFilePart in case the full size of the file is more than **10 MB** and upload.saveFilePart for smaller files.

Each call saves a portion of the data in a temporary location on the server to be used later. The storage life of each portion of data is between several minutes and several hours (depending on how busy the storage area is). After this time, the file part will become unavailable. To increase the time efficiency of a file save operation, we recommend using a call queue, so X pieces of the file are being saved at any given moment in time. Each successful operation to save a part invokes the method call to save the next part. The value of X can be tuned to achieve maximum performance.

When using one of the methods mentioned above to save file parts, one of the following data input errors may be returned:

- FILE_PARTS_INVALID - Invalid number of parts. The value is not between `1..3000`
- FILE_PART_INVALID: The file part number is invalid. The value is not between `0 and 2,999`.
- FILE_PART_TOO_BIG: The size limit (512 KB) for the content of the file part has been exceeded
- FILE_PART_EMPTY: The file part sent is empty
- FILE_PART_SIZE_INVALID - 512KB cannot be evenly divided by **part_size**
- FILE_PART_SIZE_CHANGED - The part size is different from the size of one of the previous parts in the same file

While the parts are being uploaded, an MD5 hash of the file contents can also be computed to be used later as the **md5_checksum** parameter in the inputFile constructor (since it is checked only by the server, for encrypted secret chat files it must be generated from the encrypted file).
After the entire file is successfully saved, the final method may be called and passed the generated inputFile object. In case the upload.saveBigFilePart method is used, the inputFileBig constructor must be passed, in other cases use inputFile.

The file save operation may return one of the following data input errors:

- FILE_PARTS_INVALID: The number of file parts is invalid The value is not between 1 and 3,000.
- FILE_PART_X_MISSING: Part X (where X is a number) of the file is missing from storage. Try repeating the method call to resave the part.
- MD5_CHECKSUM_INVALID: The file's checksum did not match the **md5_checksum** parameter

Re-using pre-uploaded files

```
document#9ba29cc1 flags:# id:long access_hash:long file_reference:bytes date:int mime_type:string size:int thum

---functions---

messages.getDocumentByHash#338e2464 sha256:bytes size:int mime_type:string = Document;
```

For some types of documents like GIFs, messages.getDocumentByHash can be used to search for the document on Telegram servers.
The SHA256 hash of the file is computed, and it is passed along with the file's mime type and size to the method: if the file type is correct and the file is found, a document is returned.

Downloading files

There are methods available to download files which have been successfully uploaded. The schema of the types and methods used is presented below:

```
upload.file#96a18d5 type:storage.FileType mtime:int bytes:bytes = upload.File;
upload.fileCdnRedirect#f18cda44 dc_id:int file_token:bytes encryption_key:bytes encryption_iv:bytes file_hashes

storage.fileUnknown#aa963b05 = storage.FileType;
storage.fileJpeg#7efe0e = storage.FileType;
storage.fileGif#cae1aadf = storage.FileType;
storage.filePng#a4f63c0 = storage.FileType;
storage.fileMp3#528a0677 = storage.FileType;
storage.fileMov#4b09ebbc = storage.FileType;
storage.filePartial#40bc6f52 = storage.FileType;
storage.fileMp4#b3cea0e4 = storage.FileType;
storage.fileWebp#1081464c = storage.FileType;

---functions---

upload.getFile#e3a6cfb5 location:InputFileLocation offset:int limit:int = upload.File;
```

Any file can be downloaded by calling upload.getFile.
The data for the input parameter of the InputFileLocation type is generated as follows:

```
inputFileLocation#dfdaabe1 volume_id:long local_id:int secret:long file_reference:bytes = InputFileLocation;
inputEncryptedFileLocation#f5235d55 id:long access_hash:long = InputFileLocation;
inputDocumentFileLocation#bad07584 id:long access_hash:long file_reference:bytes thumb_size:string = InputFileL
inputSecureFileLocation#cbc7ee28 id:long access_hash:long = InputFileLocation;
inputTakeoutFileLocation#29be5899 = InputFileLocation;
inputPhotoFileLocation#40181ffe id:long access_hash:long file_reference:bytes thumb_size:string = InputFileLoca
inputPeerPhotoFileLocation#27d69997 flags:# big:flags.0?true peer:InputPeer volume_id:long local_id:int = Input
inputStickerSetThumb#dbaeae9 stickerset:InputStickerSet volume_id:long local_id:int = InputFileLocation;

inputStickerSetEmpty#ffb62b95 = InputStickerSet;
inputStickerSetID#9de7a269 id:long access_hash:long = InputStickerSet;
inputStickerSetShortName#861cc8a0 short_name:string = InputStickerSet;

inputPeerSelf#7da07ec9 = InputPeer;
inputPeerChat#179be863 chat_id:int = InputPeer;
inputPeerUser#7b8e7de6 user_id:int access_hash:long = InputPeer;
inputPeerChannel#20adaef8 channel_id:int access_hash:long = InputPeer;

photo#d07504a5 flags:# has_stickers:flags.0?true id:long access_hash:long file_reference:bytes date:int sizes:V
document#9ba29cc1 flags:# id:long access_hash:long file_reference:bytes date:int mime_type:string size:int thum

photoSize#77bfb61b type:string location:FileLocation w:int h:int size:int = PhotoSize;
photoCachedSize#e9a734fa type:string location:FileLocation w:int h:int bytes:bytes = PhotoSize;
```

- For photos, inputPhotoFileLocation is used:
  - `id`, `file_reference` and `access_hash` taken from the photo constructor
  - `thumb_size` taken from the `type` field of the desired PhotoSize of the photo
- For profile pictures of users, channels, supergroups and groups, since in most occasions they are encountered as simple fileLocationToBeDeprecated constructors without an associated photo, inputPeerPhotoFileLocation has to be used:
  - `peer` is the identifier of the peer whose photo we want to download
  - `volume_id` and `local_id` are extracted from the fileLocationToBeDeprecated of the desired quality (the logic for selecting the quality of profile pictures will be changed soon)
- For documents, inputDocumentFileLocation is used:
  - `id`, `file_reference` and `access_hash` taken from the document constructor
  - If downloading the thumbnail of a document, `thumb_size` should be taken from the `type` field of the desired PhotoSize of the photo; otherwise, provide an empty string.
- For encrypted secret chat and telegram passport documents, respectively inputEncryptedFileLocation and inputSecureFileLocation have to be used, with parameters extracted from encryptedFile and secureFile (passport docs).
- For previews of sticker sets, inputStickerSetThumb is used (note: to download stickers and previews of stickers use the document method described above):
  - `stickerset` is set to the InputStickerSet constructor generated from the StickerSet
  - `volume_id` and `local_id` are extracted from the fileLocationToBeDeprecated from the `thumb` PhotoSize of the StickerSet (the logic for downloading stickerset previews will be changed soon)
- For old **deprecated** photos, if the client has cached some old fileLocations with the **deprecated** `secret` identifier, inputFileLocation is used:
  - All fields are taken from the previously cached fileLocation

The size of each file in bytes is available, which makes it possible to download the file in parts using the parameters **offset** and **limit**, similar to the way files are uploaded, also:

- The parameter **offset** must be divisible by 1 KB.
- 10485760 (1MB) must be divisible by `limit`

The file download operation may return a `FILE_REFERENCE_EXPIRED` error (or another error starting with `FILE_REFERENCE_`): in this case, the `file_reference` field of the input location must be refreshed.
The file download operation may return an upload.fileCdnRedirect constructor: in this case, these instructions must be followed for downloading CDN files.
The file download operation may also return one of the following data input errors:

- FILE_ID_INVALID: The file address is invalid
- OFFSET_INVALID: The **offset** value is invalid. The value is not divisible by 1 KB.
- LIMIT_INVALID: The **limit** value is invalid
- FILE_MIGRATE_X: The file is in Data Center No. X

Verifying downloaded chunks

```
fileHash#6242c773 offset:int limit:int hash:bytes = FileHash;

---functions---

upload.getFileHashes#c7025931 location:InputFileLocation offset:int = Vector<FileHash>;
```

In order to confirm the integrity of the downloaded file, clients are recommended to verify hashes for each downloaded part, as for CDN DCs.
upload.getFileHashes contain FileHash constructors. Each of these constructors contains the SHA-256 hash of a part of the file that starts with `offset` and takes `limit` bytes.

Before saving each portion of the data received from the DC into the file, the client can confirm that its hash matches the hash that was received from the master DC. If missing a hash for any file part, client developers must use the upload.getFileHashes method to obtain the missing hash.

Downloading webfiles

Remote HTTP files sent by inline bots in response to inline queries and in other places are represented by WebDocument constructors.
When forwarding such remote HTTP files, they should be sent using external InputMedia constructors.
Remote HTTP files can only be downloaded directly by the client if contained in a webDocumentNoProxy constructor: in this case, the file is deemed safe to download (this is the case for HTTPS files from certain trusted domains).

However, if the remote file is contained in a webDocument, to avoid leaking sensitive information the file must be downloaded through telegram's servers.
This can be done in a manner similar to normal files, with the difference that upload.getWebFile must be used, instead.

```
upload.webFile#21e753bc size:int mime_type:string file_type:storage.FileType mtime:int bytes:bytes = upload.Web

storage.fileUnknown#aa963b05 = storage.FileType;
storage.fileJpeg#7efe0e = storage.FileType;
storage.fileGif#cae1aadf = storage.FileType;
storage.filePng#a4f63c0 = storage.FileType;
storage.fileMp3#528a0677 = storage.FileType;
storage.fileMov#4b09ebbc = storage.FileType;
storage.filePartial#40bc6f52 = storage.FileType;
storage.fileMp4#b3cea0e4 = storage.FileType;
storage.fileWebp#1081464c = storage.FileType;

---functions---

upload.getWebFile#24e6818d location:InputWebFileLocation offset:int limit:int = upload.WebFile;
```

The InputWebFileLocation constructor is generated as follows.

```
inputWebFileLocation#c239d686 url:string access_hash:long = InputWebFileLocation;
inputWebFileGeoPointLocation#9f2221c9 geo_point:InputGeoPoint access_hash:long w:int h:int zoom:int scale:int =

webDocument#1c570ed1 url:string access_hash:long size:int mime_type:string attributes:Vector<DocumentAttribute>
```

```
inputGeoPoint#f3b7acc9 lat:double long:double = InputGeoPoint;

geoPoint#296f104 long:double lat:double access_hash:long = GeoPoint;
```

- inputWebFileLocation is simply generated by taking the `url` and `access_hash` fields of the webDocument constructor.
- inputWebFileGeoPointLocation is used to download a server-generated image with the map preview from a geoPoint.
  - `geo_point` is generated from the `lat` and `long` parameters of the geoPoint
  - `access_hash` is the access hash of the geoPoint
  - `w` – Map width in pixels before applying scale; 16-1024
  - `h` – Map height in pixels before applying scale; 16-1024
  - `zoom` – Map zoom level; 13-20
  - `scale` – Map scale; 1-3

General Considerations

It is recommended that large queries (upload.getFile, upload.saveFilePart) be handled through a separate session and a separate connection, in which no methods other than these should be executed. If this is done, then data transfer will cause less interference with getting updates and other method calls.

Related articles

Handling file references
How to handle file references.

Pagination in the API

Lots of Telegram API methods provide access to potentially large lists of objects, which requires pagination.

In order to fetch only relevant subset of results for each request there is a number of available input parameters. Here is a list in order how they are applied in API.

Typically, results are returned in antichronological order with descending object ID values.

`limit` parameter

A limit on the number of objects to be returned, typically between 1 and 100. When 0 is provided the limit will often default to an intermediate value like ~20.

`offset` -based pagination

For a few methods with mostly static data this parameter allows to skip `offset` elements from the beginning of list; negative values are ignored.

`offset_id` -based pagination

For most methods where results are real-time data (e.g. any chat history) `offset` value is not passed directly. Instead it is calculated from the passed `offset_id` and `add_offset` parameter values as `offsetFromID(offset_id) + add_offset`, where `offsetFromID(offset_id)` is a number of results from the beginning of list up to the result with ID `offset_id`, inclusive.

Sample use cases:

- Loading 20 messages, older than message with ID `MSGID`:

```
messages.getHistory({offset_id: MSGID, add_offset: 0, limit: 20})
```

- Loading 20 messages, newer than message with ID `MSGID`:

```
messages.getHistory({offset_id: MSGID, add_offset: -20, limit: 20})
```

- Loading 20 messages around message with ID `MSGID`:

```
messages.getHistory({offset_id: MSGID, add_offset: -10, limit: 20})
```

Additional filtering

There is a number of parameters, which are applied to the list after slicing with offset and limit, to reduce the result subset even more:

- **max_id**: Can be used to only return results with ID strictly smaller than `max_id` (e.g. message ID)
- **min_id**: Can be used to only return results with ID strictly greater than `min_id` (e.g. message ID)
- **max_date**: Can be used to only return results that are older than `max_date`.
- **min_date**: Can be used to only return results with are newer than `min_date`.
- **hash**: See below.

Hash generation

To further reduce the result subset, there is a mechanism to avoid fetching data if the resulting list hasn't changed from the one stored on client, similar to ETag.

When the client has cached results for API request, it can calculate the `hash` value for it by taking the result IDs (message IDs or other fields with name `id`) and using them to compute a 32-bit hash with the following algorithm:

```
hash = 0
for id in ids:
    hash = (((hash * 0x4F25) & 0x7FFFFFFF) + id) & 0x7FFFFFFF
```

In some cases, the result container already has a `hash` field, that can be used instead.

When the client passes a correct value, the API will return one of `*NotModified` constructors, e.g. messages.messagesNotModified instead of the actual results.

Example methods

- messages.getHistory supports all result navigation parameters including message ID hashes and except filters
- channels.getParticipants supports simple navigation using **limit** and **offset**, along with filtering and `hash` reducing using the user IDs of returned participants

End-to-End Encryption, Secret Chats

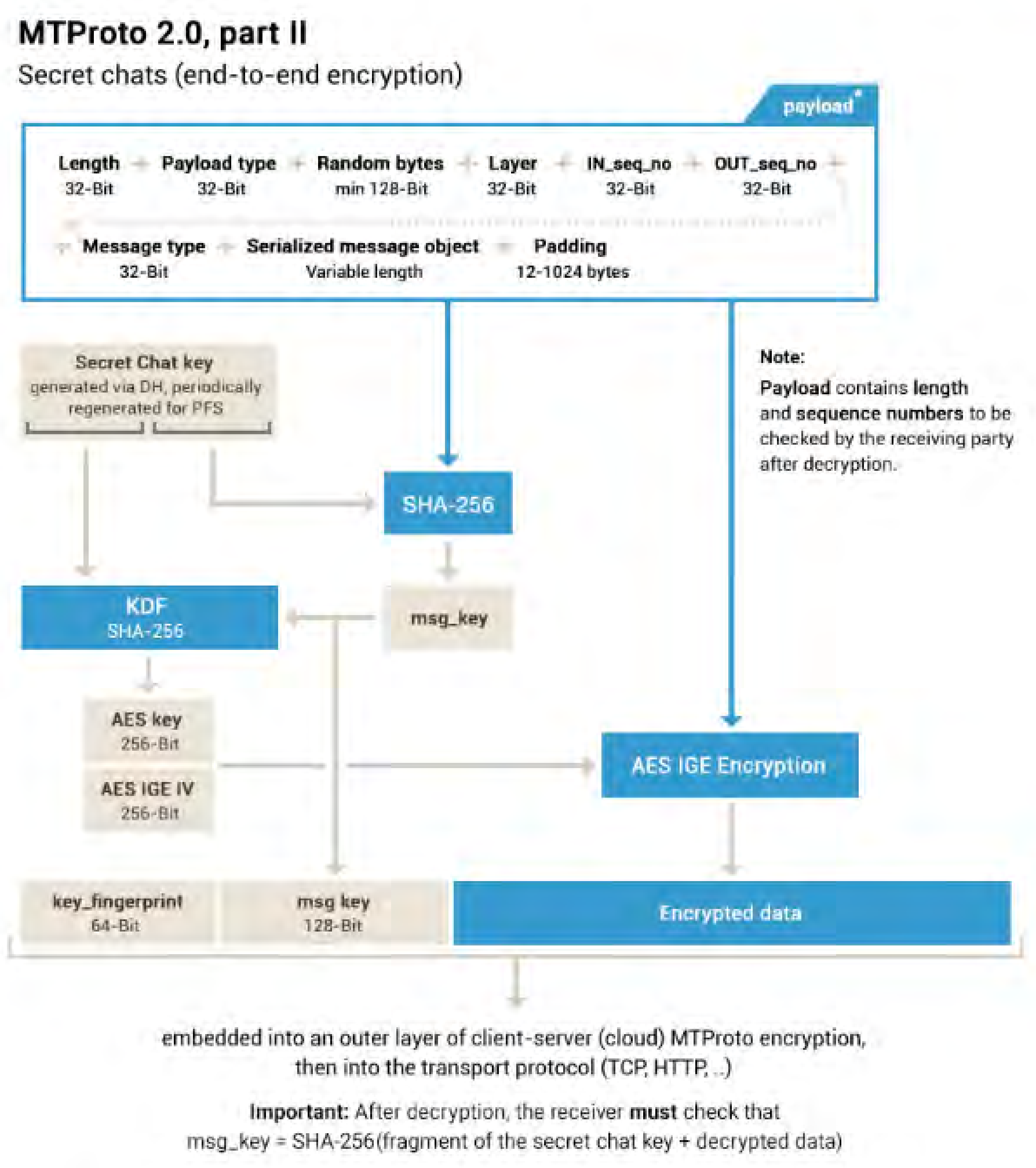
This article on MTProto's End-to-End encryption is meant for **advanced users**. If you want to learn more about Secret Chats from a less intimidating source, kindly see our general FAQ.

Note that as of version 4.6, major Telegram clients are using **MTProto 2.0**. MTProto v.1.0 is deprecated and is currently being phased out.

Related articles

Security guidelines for developers
Perfect Forward Secrecy in Secret Chats
Sequence numbers in Secret Chats
End-to-End TL Schema

Secret Chats are one-on-one chats wherein messages are encrypted with a key held only by the chat's participants. Note that the schema for these end-to-end encrypted Secret Chats is different from what is used for cloud chats:


MTProto 2.0, part II
Secret chats (end-to-end encryption)
payload*
Length 32-Bit
Payload type 32-Bit
Random bytes min 128-Bit
Layer 32-Bit
IN_seq_no 32-Bit
OUT_seq_no 32-Bit
Message type 32-Bit
Serialized message object Variable length
Padding 12-1024 bytes
Secret Chat key generated via DH, periodically regenerated for PFS
SHA-256
KDF SHA-256
msg_key
AES key 256-Bit
AES IGE IV 256-Bit
AES IGE Encryption
key_fingerprint 64-Bit
msg key 128-Bit
Encrypted data
Note: Payload contains length and sequence numbers to be checked by the receiving party after decryption.
embedded into an outer layer of client-server (cloud) MTProto encryption, then into the transport protocol (TCP, HTTP, ..)
Important: After decryption, the receiver must check that msg_key = SHA-256(fragment of the secret chat key + decrypted data)

A note on MTProto 2.0

This article describes the end-to-end encryption layer in the MTProto protocol version **2.0**. The principal differences from version 1.0 (described here for reference) are as follows:

- SHA-256 is used instead of SHA-1;
- Padding bytes are involved in the computation of msg_key;
- msg_key depends not only on the message to be encrypted, but on a portion of the secret chat key as well;
- 12..1024 padding bytes are used instead of 0..15 padding bytes in v.1.0.

See also: MTProto 2.0: Cloud Chats, server-client encryption

Key Generation

Keys are generated using the Diffie-Hellman protocol.

Let us consider the following scenario: User **A** would like to initiate end-to-end encrypted communication with User **B**.

Sending a Request

User **A** executes messages.getDhConfig to obtain the Diffie-Hellman parameters: a prime **p**, and a high order element **g**.

Executing this method before each new key generation procedure is of vital importance. It makes sense to cache the values of the parameters together with the version in order to avoid having to receive all of the values every time. If the version stored on the client is still up-to-date, the server will return the constructor messages.dhConfigNotModified.

Client is expected to check whether **p** is a safe 2048-bit prime (meaning that both **p** and **(p-1)/2** are prime, and that 2^2047 < p < 2^2048), and that **g** generates a cyclic subgroup of prime order **(p-1)/2**, i.e. is a quadratic residue **mod p**. Since **g** is always equal to 2, 3, 4, 5, 6 or 7, this is easily done using quadratic reciprocity law, yielding a simple condition on **p mod 4g** -- namely, **p mod 8 = 7** for **g = 2**; **p mod 3 = 2** for **g = 3**; no extra condition for **g = 4**; **p mod 5 = 1 or 4** for **g = 5**; **p mod 24 = 19 or 23** for **g = 6**; and **p mod 7 = 3, 5 or 6** for **g = 7**. After **g** and **p** have been checked by the client, it makes sense to cache the result, so as to avoid repeating lengthy computations in future. This cache might be shared with one used for Authorization Key generation.

If the client has an inadequate random number generator, it makes sense to pass the **random_length** parameter (random_length> 0) so the server generates its own random sequence **random** of the appropriate length. **Important**: using the server's random sequence in its raw form may be unsafe. It must be combined with a client sequence, for example, by generating a client random number of the same length (**client_random**) and using `final_random := random XOR client_random`.

Client **A** computes a 2048-bit number **a** (using sufficient entropy or the server's **random**; see above) and executes messages.requestEncryption after passing in `g_a := pow(g, a) mod dh_prime`.

User **B** receives the update updateEncryption for all associated authorization keys (all authorized devices) with the chat constructor encryptedChatRequested. The user must be shown basic information about User **A** and must be prompted to accept or reject the request.

Both clients are to check that **g**, **g_a** and **g_b** are greater than one and smaller than **p-1**. We recommend checking that **g_a** and **g_b** are between **2^{2048-64}** and **p - 2^{2048-64}** as well.

Accepting a Request

After User **B** confirms the creation of a secret chat with **A** in the client interface, Client **B** also receives up-to-date configuration parameters for the Diffie-Hellman method. Thereafter, it generates a random 2048-bit number, **b**, using rules similar to those for **a**.

Having received **g_a** from the update with encryptedChatRequested, it can immediately generate the final shared key: `key = (pow(g_a, b) mod dh_prime)`. If key length < 256 bytes, add several leading zero bytes as padding — so that the key is exactly 256 bytes long. Its fingerprint, **key_fingerprint**, is equal to the 64 last bits of SHA1 (key).

Note 1: in this particular case SHA1 is used here even for MTProto 2.0 secret chats.

Note 2: this fingerprint is used as a sanity check for the key exchange procedure to detect bugs when developing client software — it is not connected to the key visualization used on the clients as means of external authentication in secret chats. Key visualizations on the clients are generated using the first 128 bits of SHA1(intial key) followed by the first 160 bits of SHA256(key used when secret chat was updated to layer 46).

Client **B** executes messages.acceptEncryption after passing it `g_b := pow(g, b) mod dh_prime` and **key_fingerprint**.

For all of Client **B's** authorized devices, except the current one, updateEncryption updates are sent with the constructor encryptedChatDiscarded. Thereafter, the only device that will be able to access the secret chat is Device **B**, which made the call to messages.acceptEncryption.

User **A** will be sent an updateEncryption update with the constructor encryptedChat, for the authorization key that initiated the chat.

With **g_b** from the update, Client **A** can also compute the shared key `key = (pow(g_b, a) mod dh_prime)`. If key length < 256 bytes, add several leading zero bytes as padding — so that the key is exactly 256 bytes long. If the fingerprint for the received key is identical to the one that was passed to encryptedChat, incoming messages can be sent and processed. Otherwise, messages.discardEncryption must be executed and the user notified.

Perfect Forward Secrecy

In order to keep past communications safe, official Telegram clients will initiate re-keying once a key has been used to decrypt and encrypt more than 100 messages, or has been in use for more than one week, provided the key has been used to encrypt at least one message. Old keys are then securely discarded and cannot be reconstructed, even with access to the new keys currently in use.

> The re-keying protocol is further described in this article: Perfect Forward Secrecy in Secret Chats.

Please note that your client must support Forward Secrecy in Secret Chats to be compatible with official Telegram clients.

Sending and Receiving Messages in a Secret Chat

Serialization and Encryption of Outgoing Messages

A TL object of type DecryptedMessage is created and contains the message in plain text. For backward compatibility, the object must be wrapped in the constructor decryptedMessageLayer with an indication of the supported layer (starting with 46).

> The TL-Schema for the contents of end-to-end encrypted messages is available here »

The resulting construct is serialized as an array of bytes using generic TL rules. The resulting array is prepended by 4 bytes containing the array length not counting these 4 bytes.

The byte array is padded with 12 to 1024 random padding bytes to make its length divisible by 16 bytes. (In the older MTProto 1.0 encryption, only 0 to 15 padding bytes were used.)

Message key, **msg_key**, is computed as the 128 middle bits of the SHA256 of the data obtained in the previous step, prepended by 32 bytes from the shared key **key**. (For the older MTProto 1.0 encryption, **msg_key** was computed differently, as the 128 lower bits of SHA1 of the data obtained in the previous steps, *excluding the padding bytes*.)

For MTProto 2.0, the AES key **aes_key** and initialization vector **aes_iv** are computed (**key** is the shared key obtained during Key Generation) as follows:

- msg_key_large = SHA256 (substr (key, 88+x, 32) + plaintext + random_padding);
- msg_key = substr (msg_key_large, 8, 16);
- sha256_a = SHA256 (msg_key + substr (key, x, 36));
- sha256_b = SHA256 (substr (key, 40+x, 36) + msg_key);
- aes_key = substr (sha256_a, 0, 8) + substr (sha256_b, 8, 16) + substr (sha256_a, 24, 8);
- aes_iv = substr (sha256_b, 0, 8) + substr (sha256_a, 8, 16) + substr (sha256_b, 24, 8);

For MTProto 2.0, **x=0** for messages from the originator of the secret chat, **x=8** for the messages in the opposite direction.

For the obsolete MTProto 1.0, msg_key, aes_key, and aes_iv were computed differently (see this document for reference).

Data is encrypted with a 256-bit key, **aes_key**, and a 256-bit initialization vector, **aes-iv**, using AES-256 encryption with infinite garble extension (IGE). Encryption key fingerprint **key_fingerprint** and the message key **msg_key** are added at the top of the resulting byte array.

Encrypted data is embedded into a messages.sendEncrypted API call and passed to Telegram server for delivery to the other party of the Secret Chat.

Upgrading to MTProto 2.0 from MTProto 1.0

As soon as both parties in a secret chat are using at least Layer 73, they should only use MTProto 2.0 for all outgoing messages. Some of the first received messages may use MTProto 1.0, if a sufficiently high starting layer has not been negotiated during the creation of the secret chat. After the first message encrypted with MTProto 2.0 (or the first message with Layer 73 or higher) is received, all messages with higher sequence numbers must be encrypted with MTProto 2.0 as well.

As long as the current layer is lower than 73, each party should try to decrypt received messages with MTProto 1.0, and if this is not successfull (msg_key does not match), try MTProto 2.0. Once the first MTProto 2.0-encrypted message arrives (or the layer is upgraded to 73), there is no need to try MTProto 1.0 decryption for any of the further messages (unless the client is still waiting for some gaps to be closed).

Decrypting an Incoming Message

The steps above are performed in reverse order. When an encrypted message is received, you **must** check that msg_key is **in fact** equal to the 128 middle bits of the SHA256 hash of the decrypted message, prepended by 32 bytes taken from the shared **key**. If the message layer is greater than the one supported by the client, the user must be notified that the client version is out of date and prompted to update.

Sequence numbers

It is necessary to interpret all messages in their original order to protect against possible manipulations. Secret chats support a special mechanism for handling seq_no counters independently from the server.

> Proper handling of these counters is further described in this article: Sequence numbers in Secret Chats.

Please note that your client must support sequence numbers in Secret Chats to be compatible with official Telegram clients.

Sending Encrypted Files

All files sent to secret chats are encrypted with one-time keys that are in no way related to the chat's shared key. Before an encrypted file is sent, it is assumed that the encrypted file's address will be attached to the outside of an encrypted message using the **file** parameter of the messages.sendEncryptedFile method and that the key for direct decryption will be sent in the body of the message (the **key** parameter in the constructors decryptedMessageMediaPhoto, decryptedMessageMediaVideo and decryptedMessageMediaFile.

Prior to a file being sent to a secret chat, 2 random 256-bit numbers are computed which will serve as the AES key and initialization vector used to encrypt the file. AES-256 encryption with infinite garble extension (IGE) is used in like manner.

The key fingerprint is computed as follows:

- digest = md5(key + iv)
- fingerprint = substr(digest, 0, 4) XOR substr(digest, 4, 4)

The encrypted contents of a file are stored on the server in much the same way as those of a file in cloud chats: piece by piece using calls to upload.saveFilePart. A subsequent call to messages.sendEncryptedFile will assign an identifier to the stored file and send the address together with the message. The recipient will receive an update with encryptedMessage, and the **file** parameter will contain file information.

Incoming and outgoing encrypted files can be forwarded to other secret chats using the constructor inputEncryptedFile to avoid saving the same content on the server twice.

Working with an Update Box

Secret chats are associated with specific devices (or rather with authorization keys), not users. A conventional message box, which uses **pts** to describe the client's status, is not suitable, because it is designed for long-term message storage and message access from different devices.

An additional temporary message queue is introduced as a solution to this problem. When an update regarding a message from a secret chat is sent, a new value of **qts** is sent, which helps reconstruct the difference if there has been a long break in the connection or in case of loss of an update.

As the number of events increases, the value of **qts** increases by 1 with each new event. The initial value may not (and will not) be equal to 0.

The fact that events from the temporary queue have been received and stored by the client is acknowledged explicitly by a call to the messages.receivedQueue method or implicitly by a call to updates.getDifference (the value of **qts** passed, not the final state). All messages acknowledged as delivered by the client, as well as any messages older than 7 days, may (and will) be deleted from the server.

Upon de-authorization, the event queue of the corresponding device will be forcibly cleared, and the value of **qts** will become irrelevant.

Updating to new layers

Your client should always store the maximal layer that is known to be supported by the client on the other side of a secret chat. When the secret chat is first created, this value should be initialized to 46. This remote layer value must always be updated immediately after receiving *any* packet containing information of an upper layer, i.e.:

- any secret chat message containing *layer_no* in its `decryptedMessageLayer` with *layer*>=46, or
- a decryptedMessageActionNotifyLayer service message, wrapped as if it were the decryptedMessageService constructor of the obsolete layer 8 (constructor `decryptedMessageService#aa48327d`).

Notifying the remote client about your local layer

In order to notify the remote client of your local layer, your client must send a message of the `decryptedMessageActionNotifyLayer` type. This notification must be wrapped in a constructor of an appropriate layer.

There are two cases when your client must notify the remote client about its local layer:

1. As soon as a new secret chat has been created, immediately after the secret key has been successfully exchanged.
2. Immediately after the local client has been updated to support a new secret chat layer. In this case notifications must be sent to **all** currently existing secret chats. Note that this is only necessary when updating to new layers that contain changes in the secret chats implementation (e.g. you don't need to do this when your client is updated from Layer 46 to Layer 47).

Security Guidelines for Client Developers

See also:

Perfect Forward Secrecy
Secret chats, end-to-end encryption
Perfect Forward Secrecy in Secret Chats
MTProto 2.0, Detailed Description

While MTProto is designed to be a reasonably fast and secure protocol, its advantages can be easily negated by careless implementation. We collected some security guidelines for client software developers on this page. All Telegram clients are required to comply.

> Note that as of version 4.6, major Telegram clients are using **MTProto 2.0**.
> MTProto v.1.0 is deprecated and is currently being phased out.

Diffie—Hellman key exchange

We use DH key exchange in two cases:

- Creating an authorization key
- Establishing Secret Chats with end-to-end encryption

In both cases, there are some verifications to be done whenever DH is used:

Validation of DH parameters

Client is expected to check whether **p = dh_prime** is a safe 2048-bit prime (meaning that both **p** and **(p-1)/2** are prime, and that $2^{2047 < p < 2}2048$), and that **g** generates a cyclic subgroup of prime order **(p-1)/2**, i.e. is a quadratic residue **mod p**. Since **g** is always equal to 2, 3, 4, 5, 6 or 7, this is easily done using quadratic reciprocity law, yielding a simple condition on **p mod 4g** — namely, **p mod 8 = 7** for **g = 2**; **p mod 3 = 2** for **g = 3**; no extra condition for **g = 4**; **p mod 5 = 1 or 4** for **g = 5**; **p mod 24 = 19 or 23** for **g = 6**; and **p mod 7 = 3, 5 or 6** for **g = 7**. After **g** and **p** have been checked by the client, it makes sense to cache the result, so as not to repeat lengthy computations in future.

If the verification takes too long (which is the case for older mobile devices), one might initially run only 15 Miller—Rabin iterations (use parameter 30 in Java) for verifying primeness of **p** and **(p - 1)/2** with error probability not exceeding one billionth, and do more iterations in the background later.

Another way to optimize this is to embed into the client application code a small table with some known "good" couples **(g,p)** (or just known safe primes **p**, since the condition on **g** is easily verified during execution), checked during code generation phase, so as to avoid doing such verification during runtime altogether. The server rarely changes these values, thus one usually needs to put the current value of server's **dh_prime** into such a table. For example, the current value of **dh_prime** equals (in big-endian byte order)

```
C7 1C AE B9 C6 B1 C9 04 8E 6C 52 2F 70 F1 3F 73 98 0D 40 23 8E 3E 21 C1 49 34 D0 37 56 3D 93 0F 48 19 8A 0A A7
```

g_a and g_b validation

Apart from the conditions on the Diffie-Hellman prime **dh_prime** and generator **g**, both sides are to check that **g**, **g_a** and **g_b** are greater than **1** and less than **dh_prime - 1**. We recommend checking that **g_a** and **g_b** are between **2^{2048-64}** and **dh_prime - 2^{2048-64}** as well.

Checking SHA1 hash values during key generation

Once the client receives a `server_DH_params_ok` answer in step 5) of the Authorization Key generation protocol and decrypts it obtaining `answer_with_hash`, it MUST check that

```
answer_with_hash := SHA1(answer) + answer + (0-15 random bytes)
```

In other words, the first 20 bytes of `answer_with_hash` must be equal to SHA1 of the remainder of the decrypted message without the padding random bytes.

Checking nonce, server_nonce and new_nonce fields

When the client receives and/or decrypts server messages during creation of Authorization Key, and these messages contain some nonce fields already known to the client from messages previously obtained during the same run of the protocol, the client is to check that these fields indeed contain the values previously known.

Using secure pseudorandom number generator to create DH secret parameters `a` and `b`

Client must use a cryptographically secure PRNG to generate secret exponents `a` or `b` for DH key exchange. For secret chats, the client might request some entropy (random bytes) from the server while invoking messages.getDhConfig and feed these random bytes into its PRNG (for example, by `PRNG_seed` if OpenSSL library is used), but never using these "random" bytes by themselves or replacing by them the local PRNG seed. One should mix bytes received from server into local PRNG seed.

MTProto Encrypted Messages

Some important checks are to be done while sending and especially receiving encrypted MTProto messages.

Checking SHA256 hash value of msg_key

`msg_key` is used not only to compute the AES key and IV to decrypt the received message. After decryption, the client **MUST** check that `msg_key` is indeed equal to SHA256 of the plaintext obtained as the result of decryption (including the final 12…1024 padding bytes), prepended with 32 bytes taken from the `auth_key`, as explained in MTProto 2.0 Description.

If an error is encountered before this check could be performed, the client **must** perform the `msg_key` check anyway before returning any result. Note that the response to any error encountered before the `msg_key` check **must** be the same as the response to a failed `msg_key` check.

Checking message length

The client **must** check that the length of the message or container obtained from the decrypted message (computed from its `length` field) does not exceed the total size of the plaintext, and that the difference (i.e. the length of the random padding) lies in the range from 12 to 1024 bytes.

The length should be always divisible by 4 and non-negative. On no account the client is to access data past the end of the decryption buffer containing the plaintext message.

Checking session_id

The client is to check that the `session_id` field in the decrypted message indeed equals to that of an active session created by the client.

Checking msg_id

The client must check that `msg_id` has even parity for messages from client to server, and odd parity for messages from server to client.

In addition, the identifiers (msg_id) of the last N messages received from the other side must be stored, and if a message comes in with an msg_id lower than all or equal to any of the stored values, that message is to be ignored. Otherwise, the new message msg_id is added to the set, and, if the number of stored msg_id values is greater than N, the oldest (i. e. the lowest) is discarded.

In addition, msg_id values that belong over 30 seconds in the future or over 300 seconds in the past are to be ignored (recall that `msg_id` approximately equals unixtime * 2^32). This is especially important for the server. The client would also find this useful (to protect from a replay attack), but only if it is certain of its time (for example, if its time has been synchronized with that of the server).

Certain client-to-server service messages containing data sent by the client to the server (for example, `msg_id` of a recent client query) may, nonetheless, be processed on the client even if the time appears to be "incorrect". This is especially true of messages to change server_salt and notifications about invalid time on the client. See Mobile Protocol: Service Messages.

Behavior in case of mismatch

If one of the checks listed above fails, the client is to completely discard the message obtained from server. We also recommend closing and reestablishing the TCP connection to the server, then retrying the operation or the whole key generation protocol.

No information from incorrect messages can be used. Even if the application throws an exception and dies, this is much better than continuing with invalid data.

Notice that invalid messages will infrequently appear during normal work even if no malicious tampering is being done. This is due to network transmission errors. We recommend ignoring the invalid message and closing the TCP connection, then creating a new TCP connection to the server and retrying the original query.

> The previous version of security recommendations relevant for MTProto 1.0 clients is available here.

Schema

Below you will find the current TL-schema. More details on TL »

See also the detailed schema in JSON »

See also TL-Schema for end-to-end encrypted messages »

Layer 105

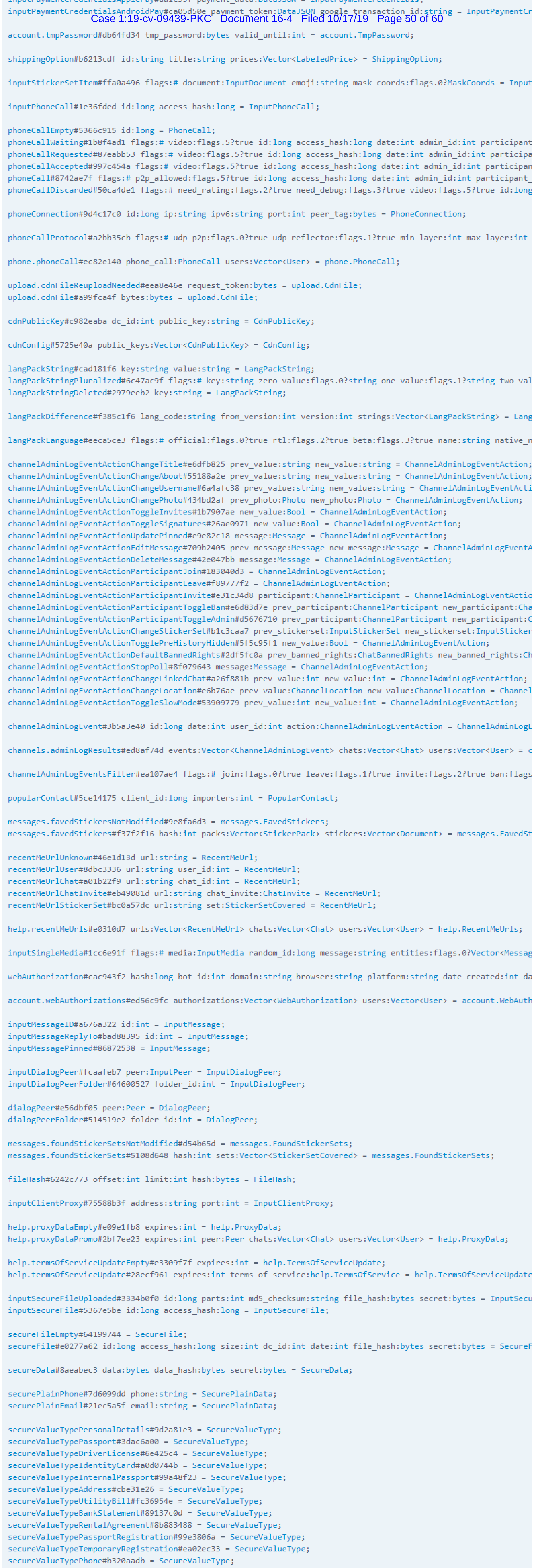
```
boolFalse#bc799737 = Bool;
boolTrue#997275b5 = Bool;

true#3fedd339 = True;

vector#1cb5c415 {t:Type} # [ t ] = Vector t;

error#c4b9f9bb code:int text:string = Error;

null#56730bcc = Null;

inputPeerEmpty#7f3b18ea = InputPeer;
inputPeerSelf#7da07ec9 = InputPeer;
inputPeerChat#179be863 chat_id:int = InputPeer;
inputPeerUser#7b8e7de6 user_id:int access_hash:long = InputPeer;
inputPeerChannel#20adaef8 channel_id:int access_hash:long = InputPeer;
inputPeerUserFromMessage#17bae2e6 peer:InputPeer msg_id:int user_id:int = InputPeer;
inputPeerChannelFromMessage#9c95f7bb peer:InputPeer msg_id:int channel_id:int = InputPeer;

inputUserEmpty#b98886cf = InputUser;
inputUserSelf#f7c1b13f = InputUser;
inputUser#d8292816 user_id:int access_hash:long = InputUser;
inputUserFromMessage#2d117597 peer:InputPeer msg_id:int user_id:int = InputUser;

inputPhoneContact#f392b7f4 client_id:long phone:string first_name:string last_name:string = InputContact;

inputFile#f52ff27f id:long parts:int name:string md5_checksum:string = InputFile;
inputFileBig#fa4f0bb5 id:long parts:int name:string = InputFile;

inputMediaEmpty#9664f57f = InputMedia;
inputMediaUploadedPhoto#1e287d04 flags:# file:InputFile stickers:flags.0?Vector<InputDocument> ttl_seconds:flag
inputMediaPhoto#b3ba0635 flags:# id:InputPhoto ttl_seconds:flags.0?int = InputMedia;
inputMediaGeoPoint#f9c44144 geo_point:InputGeoPoint = InputMedia;
inputMediaContact#f8ab7dfb phone_number:string first_name:string last_name:string vcard:string = InputMedia;
inputMediaUploadedDocument#5b38c6c1 flags:# nosound_video:flags.3?true file:InputFile thumb:flags.2?InputFile m
inputMediaDocument#23ab23d2 flags:# id:InputDocument ttl_seconds:flags.0?int = InputMedia;
inputMediaVenue#c13d1c11 geo_point:InputGeoPoint title:string address:string provider:string venue_id:string ve
inputMediaGifExternal#4843b0fd url:string q:string = InputMedia;
inputMediaPhotoExternal#e5bbfe1a flags:# url:string ttl_seconds:flags.0?int = InputMedia;
inputMediaDocumentExternal#fb52dc99 flags:# url:string ttl_seconds:flags.0?int = InputMedia;
inputMediaGame#d33f43f3 id:InputGame = InputMedia;
inputMediaInvoice#f4e096c3 flags:# title:string description:string photo:flags.0?InputWebDocument invoice:Invoi
inputMediaGeoLive#ce4e82fd flags:# stopped:flags.0?true geo_point:InputGeoPoint period:flags.1?int = InputMedia
inputMediaPoll#6b3765b poll:Poll = InputMedia;

inputChatPhotoEmpty#1ca48f57 = InputChatPhoto;
inputChatUploadedPhoto#927c55b4 file:InputFile = InputChatPhoto;
inputChatPhoto#8953ad37 id:InputPhoto = InputChatPhoto;

inputGeoPointEmpty#e4c123d6 = InputGeoPoint;
inputGeoPoint#f3b7acc9 lat:double long:double = InputGeoPoint;

inputPhotoEmpty#1cd7bf0d = InputPhoto;
inputPhoto#3bb3b94a id:long access_hash:long file_reference:bytes = InputPhoto;

inputFileLocation#dfdaabe1 volume_id:long local_id:int secret:long file_reference:bytes = InputFileLocation;
inputEncryptedFileLocation#f5235d55 id:long access_hash:long = InputFileLocation;
inputDocumentFileLocation#bad07584 id:long access_hash:long file_reference:bytes thumb_size:string = InputFileL
inputSecureFileLocation#cbc7ee28 id:long access_hash:long = InputFileLocation;
inputTakeoutFileLocation#29be5899 = InputFileLocation;
inputPhotoFileLocation#40181ffe id:long access_hash:long file_reference:bytes thumb_size:string = InputFileLoca
inputPeerPhotoFileLocation#27d69997 flags:# big:flags.0?true peer:InputPeer volume_id:long local_id:int = Input
inputStickerSetThumb#dbaeae9 stickerset:InputStickerSet volume_id:long local_id:int = InputFileLocation;

peerUser#9db1bc6d user_id:int = Peer;
peerChat#bad0e5bb chat_id:int = Peer;
peerChannel#bddde532 channel_id:int = Peer;

storage.fileUnknown#aa963b05 = storage.FileType;
storage.filePartial#40bc6f52 = storage.FileType;
storage.fileJpeg#7efe0e = storage.FileType;
storage.fileGif#cae1aadf = storage.FileType;
storage.filePng#a4f63c0 = storage.FileType;
storage.filePdf#ae1e508d = storage.FileType;
storage.fileMp3#528a0677 = storage.FileType;
storage.fileMov#4b09ebbc = storage.FileType;
storage.fileMp4#b3cea0e4 = storage.FileType;
storage.fileWebp#1081464c = storage.FileType;

userEmpty#200250ba id:int = User;
user#938458c1 flags:# self:flags.10?true contact:flags.11?true mutual_contact:flags.12?true deleted:flags.13?tr

userProfilePhotoEmpty#4f11bae1 = UserProfilePhoto;
userProfilePhoto#ecd75d8c photo_id:long photo_small:FileLocation photo_big:FileLocation dc_id:int = UserProfile

userStatusEmpty#9d05049 = UserStatus;
userStatusOnline#edb93949 expires:int = UserStatus;
userStatusOffline#8c703f was_online:int = UserStatus;
userStatusRecently#e26f42f1 = UserStatus;
userStatusLastWeek#7bf09fc = UserStatus;
userStatusLastMonth#77ebc742 = UserStatus;

chatEmpty#9ba2d800 id:int = Chat;
chat#3bda1bde flags:# creator:flags.0?true kicked:flags.1?true left:flags.2?true deactivated:flags.5?true id:in
chatForbidden#7328bdb id:int title:string = Chat;
channel#d31a961e flags:# creator:flags.0?true left:flags.2?true broadcast:flags.5?true verified:flags.7?true me
channelForbidden#289da732 flags:# broadcast:flags.5?true megagroup:flags.8?true id:int access_hash:long title:s

chatFull#1b7c9db3 flags:# can_set_username:flags.7?true has_scheduled:flags.8?true id:int about:string particip
channelFull#2d895c74 flags:# can_view_participants:flags.3?true can_set_username:flags.6?true can_set_stickers:

chatParticipant#c8d7493e user_id:int inviter_id:int date:int = ChatParticipant;
chatParticipantCreator#da13538a user_id:int = ChatParticipant;
chatParticipantAdmin#e2d6e436 user_id:int inviter_id:int date:int = ChatParticipant;

chatParticipantsForbidden#fc900c2b flags:# chat_id:int self_participant:flags.0?ChatParticipant = ChatParticipa
chatParticipants#3f460fed chat_id:int participants:Vector<ChatParticipant> version:int = ChatParticipants;

chatPhotoEmpty#37c1011c = ChatPhoto;
chatPhoto#475cdbd5 photo_small:FileLocation photo_big:FileLocation dc_id:int = ChatPhoto;
```

```
messageEmpty#83e5de54 id:int = Message;
message#452c0e65 flags:# out:flags.1?true mentioned:flags.4?true media_unread:flags.5?true silent:flags.13?true
messageService#9e19a1f6 flags:# out:flags.1?true mentioned:flags.4?true media_unread:flags.5?true silent:flags.

messageMediaEmpty#3ded6320 = MessageMedia;
messageMediaPhoto#695150d7 flags:# photo:flags.0?Photo ttl_seconds:flags.2?int = MessageMedia;
messageMediaGeo#56e0d474 geo:GeoPoint = MessageMedia;
messageMediaContact#cbf24940 phone_number:string first_name:string last_name:string vcard:string user_id:int =
messageMediaUnsupported#9f84f49e = MessageMedia;
messageMediaDocument#9cb070d7 flags:# document:flags.0?Document ttl_seconds:flags.2?int = MessageMedia;
messageMediaWebPage#a32dd600 webpage:WebPage = MessageMedia;
messageMediaVenue#2ec0533f geo:GeoPoint title:string address:string provider:string venue_id:string venue_type:
messageMediaGame#fdb19008 game:Game = MessageMedia;
messageMediaInvoice#84551347 flags:# shipping_address_requested:flags.1?true test:flags.3?true title:string des
messageMediaGeoLive#7c3c2609 geo:GeoPoint period:int = MessageMedia;
messageMediaPoll#4bd6e798 poll:Poll results:PollResults = MessageMedia;

messageActionEmpty#b6aef7b0 = MessageAction;
messageActionChatCreate#a6638b9a title:string users:Vector<int> = MessageAction;
messageActionChatEditTitle#b5a1ce5a title:string = MessageAction;
messageActionChatEditPhoto#7fcb13a8 photo:Photo = MessageAction;
messageActionChatDeletePhoto#95e3fbef = MessageAction;
messageActionChatAddUser#488a7337 users:Vector<int> = MessageAction;
messageActionChatDeleteUser#b2ae9b0c user_id:int = MessageAction;
messageActionChatJoinedByLink#f89cf5e8 inviter_id:int = MessageAction;
messageActionChannelCreate#95d2ac92 title:string = MessageAction;
messageActionChatMigrateTo#51bdb021 channel_id:int = MessageAction;
messageActionChannelMigrateFrom#b055eaee title:string chat_id:int = MessageAction;
messageActionPinMessage#94bd38ed = MessageAction;
messageActionHistoryClear#9fbab604 = MessageAction;
messageActionGameScore#92a72876 game_id:long score:int = MessageAction;
messageActionPaymentSentMe#8f31b327 flags:# currency:string total_amount:long payload:bytes info:flags.0?Paymen
messageActionPaymentSent#40699cd0 currency:string total_amount:long = MessageAction;
messageActionPhoneCall#80e11a7f flags:# video:flags.2?true call_id:long reason:flags.0?PhoneCallDiscardReason d
messageActionScreenshotTaken#4792929b = MessageAction;
messageActionCustomAction#fae69f56 message:string = MessageAction;
messageActionBotAllowed#abe9affe domain:string = MessageAction;
messageActionSecureValuesSentMe#1b287353 values:Vector<SecureValue> credentials:SecureCredentialsEncrypted = Me
messageActionSecureValuesSent#d95c6154 types:Vector<SecureValueType> = MessageAction;
messageActionContactSignUp#f3f25f76 = MessageAction;

dialog#2c171f72 flags:# pinned:flags.2?true unread_mark:flags.3?true peer:Peer top_message:int read_inbox_max_i
dialogFolder#71bd134c flags:# pinned:flags.2?true folder:Folder peer:Peer top_message:int unread_muted_peers_co

photoEmpty#2331b22d id:long = Photo;
photo#d07504a5 flags:# has_stickers:flags.0?true id:long access_hash:long file_reference:bytes date:int sizes:V

photoSizeEmpty#e17e23c type:string = PhotoSize;
photoSize#77bfb61b type:string location:FileLocation w:int h:int size:int = PhotoSize;
photoCachedSize#e9a734fa type:string location:FileLocation w:int h:int bytes:bytes = PhotoSize;
photoStrippedSize#e0b0bc2e type:string bytes:bytes = PhotoSize;

geoPointEmpty#1117dd5f = GeoPoint;
geoPoint#296f104 long:double lat:double access_hash:long = GeoPoint;

auth.sentCode#5e002502 flags:# type:auth.SentCodeType phone_code_hash:string next_type:flags.1?auth.CodeType ti

auth.authorization#cd050916 flags:# tmp_sessions:flags.0?int user:User = auth.Authorization;
auth.authorizationSignUpRequired#44747e9a flags:# terms_of_service:flags.0?help.TermsOfService = auth.Authoriza

auth.exportedAuthorization#df969c2d id:int bytes:bytes = auth.ExportedAuthorization;

inputNotifyPeer#b8bc5b0c peer:InputPeer = InputNotifyPeer;
inputNotifyUsers#193b4417 = InputNotifyPeer;
inputNotifyChats#4a95e84e = InputNotifyPeer;
inputNotifyBroadcasts#b1db7c7e = InputNotifyPeer;

inputPeerNotifySettings#9c3d198e flags:# show_previews:flags.0?Bool silent:flags.1?Bool mute_until:flags.2?int

peerNotifySettings#af509d20 flags:# show_previews:flags.0?Bool silent:flags.1?Bool mute_until:flags.2?int sound

peerSettings#818426cd flags:# report_spam:flags.0?true add_contact:flags.1?true block_contact:flags.2?true shar

wallPaper#a437c3ed id:long flags:# creator:flags.0?true default:flags.1?true pattern:flags.3?true dark:flags.4?

inputReportReasonSpam#58dbcab8 = ReportReason;
inputReportReasonViolence#1e22c78d = ReportReason;
inputReportReasonPornography#2e59d922 = ReportReason;
inputReportReasonChildAbuse#adf44ee3 = ReportReason;
inputReportReasonOther#e1746d0a text:string = ReportReason;
inputReportReasonCopyright#9b89f93a = ReportReason;
inputReportReasonGeoIrrelevant#dbd4feed = ReportReason;

userFull#edf17c12 flags:# blocked:flags.0?true phone_calls_available:flags.4?true phone_calls_private:flags.5?t

contact#f911c994 user_id:int mutual:Bool = Contact;

importedContact#d0028438 user_id:int client_id:long = ImportedContact;

contactBlocked#561bc879 user_id:int date:int = ContactBlocked;

contactStatus#d3680c61 user_id:int status:UserStatus = ContactStatus;

contacts.contactsNotModified#b74ba9d2 = contacts.Contacts;
contacts.contacts#eae87e42 contacts:Vector<Contact> saved_count:int users:Vector<User> = contacts.Contacts;

contacts.importedContacts#77d01c3b imported:Vector<ImportedContact> popular_invites:Vector<PopularContact> retr

contacts.blocked#1c138d15 blocked:Vector<ContactBlocked> users:Vector<User> = contacts.Blocked;
contacts.blockedSlice#900802a1 count:int blocked:Vector<ContactBlocked> users:Vector<User> = contacts.Blocked;

messages.dialogs#15ba6c40 dialogs:Vector<Dialog> messages:Vector<Message> chats:Vector<Chat> users:Vector<User>
messages.dialogsSlice#71e094f3 count:int dialogs:Vector<Dialog> messages:Vector<Message> chats:Vector<Chat> use
messages.dialogsNotModified#f0e3e596 count:int = messages.Dialogs;

messages.messages#8c718e87 messages:Vector<Message> chats:Vector<Chat> users:Vector<User> = messages.Messages;
messages.messagesSlice#c8edce1e flags:# inexact:flags.1?true count:int next_rate:flags.0?int messages:Vector<Me
messages.channelMessages#99262e37 flags:# inexact:flags.1?true pts:int count:int messages:Vector<Message> chats
messages.messagesNotModified#74535f21 count:int = messages.Messages;

messages.chats#64ff9fd5 chats:Vector<Chat> = messages.Chats;
messages.chatsSlice#9cd81144 count:int chats:Vector<Chat> = messages.Chats;

messages.chatFull#e5d7d19c full_chat:ChatFull chats:Vector<Chat> users:Vector<User> = messages.ChatFull;

messages.affectedHistory#b45c69d1 pts:int pts_count:int offset:int = messages.AffectedHistory;

inputMessagesFilterEmpty#57e2f66c = MessagesFilter;
inputMessagesFilterPhotos#9609a51c = MessagesFilter;
inputMessagesFilterVideo#9fc00e65 = MessagesFilter;
inputMessagesFilterPhotoVideo#56e9f0e4 = MessagesFilter;
inputMessagesFilterDocument#9eddf188 = MessagesFilter;
inputMessagesFilterUrl#7ef0dd87 = MessagesFilter;
inputMessagesFilterGif#ffc86587 = MessagesFilter;
inputMessagesFilterVoice#50f5c392 = MessagesFilter;
inputMessagesFilterMusic#3751b49e = MessagesFilter;
inputMessagesFilterChatPhotos#3a20ecb8 = MessagesFilter;
inputMessagesFilterPhoneCalls#80c99768 flags:# missed:flags.0?true = MessagesFilter;
inputMessagesFilterRoundVoice#7a7c17a4 = MessagesFilter;
inputMessagesFilterRoundVideo#b549da53 = MessagesFilter;
inputMessagesFilterMyMentions#c1f8e69a = MessagesFilter;
```

```
inputMessagesFilterGeo#e7026d0d = MessagesFilter;
inputMessagesFilterContacts#e062db83 = MessagesFilter;

updateNewMessage#1f2b0afd message:Message pts:int pts_count:int = Update;
updateMessageID#4e90bfd6 id:int random_id:long = Update;
updateDeleteMessages#a20db0e5 messages:Vector<int> pts:int pts_count:int = Update;
updateUserTyping#5c486927 user_id:int action:SendMessageAction = Update;
updateChatUserTyping#9a65ea1f chat_id:int user_id:int action:SendMessageAction = Update;
updateChatParticipants#7761198 participants:ChatParticipants = Update;
updateUserStatus#1bfbd823 user_id:int status:UserStatus = Update;
updateUserName#a7332b73 user_id:int first_name:string last_name:string username:string = Update;
updateUserPhoto#95313b0c user_id:int date:int photo:UserProfilePhoto previous:Bool = Update;
updateNewEncryptedMessage#12bcbd9a message:EncryptedMessage qts:int = Update;
updateEncryptedChatTyping#1710f156 chat_id:int = Update;
updateEncryption#b4a2e88d chat:EncryptedChat date:int = Update;
updateEncryptedMessagesRead#38fe25b7 chat_id:int max_date:int date:int = Update;
updateChatParticipantAdd#ea4b0e5c chat_id:int user_id:int inviter_id:int date:int version:int = Update;
updateChatParticipantDelete#6e5f8c22 chat_id:int user_id:int version:int = Update;
updateDcOptions#8e5e9873 dc_options:Vector<DcOption> = Update;
updateUserBlocked#80ece81a user_id:int blocked:Bool = Update;
updateNotifySettings#bec268ef peer:NotifyPeer notify_settings:PeerNotifySettings = Update;
updateServiceNotification#ebe46819 flags:# popup:flags.0?true inbox_date:flags.1?int type:string message:string
updatePrivacy#ee3b272a key:PrivacyKey rules:Vector<PrivacyRule> = Update;
updateUserPhone#12b9417b user_id:int phone:string = Update;
updateReadHistoryInbox#9c974fdf flags:# folder_id:flags.0?int peer:Peer max_id:int still_unread_count:int pts:i
updateReadHistoryOutbox#2f2f21bf peer:Peer max_id:int pts:int pts_count:int = Update;
updateWebPage#7f891213 webpage:WebPage pts:int pts_count:int = Update;
updateReadMessagesContents#68c13933 messages:Vector<int> pts:int pts_count:int = Update;
updateChannelTooLong#eb0467fb flags:# channel_id:int pts:flags.0?int = Update;
updateChannel#b6d45656 channel_id:int = Update;
updateNewChannelMessage#62ba04d9 message:Message pts:int pts_count:int = Update;
updateReadChannelInbox#330b5424 flags:# folder_id:flags.0?int channel_id:int max_id:int still_unread_count:int
updateDeleteChannelMessages#c37521c9 channel_id:int messages:Vector<int> pts:int pts_count:int = Update;
updateChannelMessageViews#98a12b4b channel_id:int id:int views:int = Update;
updateChatParticipantAdmin#b6901959 chat_id:int user_id:int is_admin:Bool version:int = Update;
updateNewStickerSet#688a30aa stickerset:messages.StickerSet = Update;
updateStickerSetsOrder#bb2d201 flags:# masks:flags.0?true order:Vector<long> = Update;
updateStickerSets#43ae3dec = Update;
updateSavedGifs#9375341e = Update;
updateBotInlineQuery#54826690 flags:# query_id:long user_id:int query:string geo:flags.0?GeoPoint offset:string
updateBotInlineSend#e48f964 flags:# user_id:int query:string geo:flags.0?GeoPoint id:string msg_id:flags.1?Inpu
updateEditChannelMessage#1b3f4df7 message:Message pts:int pts_count:int = Update;
updateChannelPinnedMessage#98592475 channel_id:int id:int = Update;
updateBotCallbackQuery#e73547e1 flags:# query_id:long user_id:int peer:Peer msg_id:int chat_instance:long data:
updateEditMessage#e40370a3 message:Message pts:int pts_count:int = Update;
updateInlineBotCallbackQuery#f9d27a5a flags:# query_id:long user_id:int msg_id:InputBotInlineMessageID chat_ins
updateReadChannelOutbox#25d6c9c7 channel_id:int max_id:int = Update;
updateDraftMessage#ee2bb969 peer:Peer draft:DraftMessage = Update;
updateReadFeaturedStickers#571d2742 = Update;
updateRecentStickers#9a422c20 = Update;
updateConfig#a229dd06 = Update;
updatePtsChanged#3354678f = Update;
updateChannelWebPage#40771900 channel_id:int webpage:WebPage pts:int pts_count:int = Update;
updateDialogPinned#6e6fe51c flags:# pinned:flags.0?true folder_id:flags.1?int peer:DialogPeer = Update;
updatePinnedDialogs#fa0f3ca2 flags:# folder_id:flags.1?int order:flags.0?Vector<DialogPeer> = Update;
updateBotWebhookJSON#8317c0c3 data:DataJSON = Update;
updateBotWebhookJSONQuery#9b9240a6 query_id:long data:DataJSON timeout:int = Update;
updateBotShippingQuery#e0cdc940 query_id:long user_id:int payload:bytes shipping_address:PostAddress = Update;
updateBotPrecheckoutQuery#5d2f3aa9 flags:# query_id:long user_id:int payload:bytes info:flags.0?PaymentRequeste
updatePhoneCall#ab0f6b1e phone_call:PhoneCall = Update;
updateLangPackTooLong#46560264 lang_code:string = Update;
updateLangPack#56022f4d difference:LangPackDifference = Update;
updateFavedStickers#e511996d = Update;
updateChannelReadMessagesContents#89893b45 channel_id:int messages:Vector<int> = Update;
updateContactsReset#7084a7be = Update;
updateChannelAvailableMessages#70db6837 channel_id:int available_min_id:int = Update;
updateDialogUnreadMark#e16459c3 flags:# unread:flags.0?true peer:DialogPeer = Update;
updateUserPinnedMessage#4c43da18 user_id:int id:int = Update;
updateChatPinnedMessage#e10db349 chat_id:int id:int version:int = Update;
updateMessagePoll#aca1657b flags:# poll_id:long poll:flags.0?Poll results:PollResults = Update;
updateChatDefaultBannedRights#54c01850 peer:Peer default_banned_rights:ChatBannedRights version:int = Update;
updateFolderPeers#19360dc0 folder_peers:Vector<FolderPeer> pts:int pts_count:int = Update;
updatePeerSettings#6a7e7366 peer:Peer settings:PeerSettings = Update;
updatePeerLocated#b4afcfb0 peers:Vector<PeerLocated> = Update;
updateNewScheduledMessage#39a51dfb message:Message = Update;
updateDeleteScheduledMessages#90866cee peer:Peer messages:Vector<int> = Update;
updateTheme#8216fba3 theme:Theme = Update;

updates.state#a56c2a3e pts:int qts:int date:int seq:int unread_count:int = updates.State;

updates.differenceEmpty#5d75a138 date:int seq:int = updates.Difference;
updates.difference#f49ca0 new_messages:Vector<Message> new_encrypted_messages:Vector<EncryptedMessage> other_up
updates.differenceSlice#a8fb1981 new_messages:Vector<Message> new_encrypted_messages:Vector<EncryptedMessage> o
updates.differenceTooLong#4afe8f6d pts:int = updates.Difference;

updatesTooLong#e317af7e = Updates;
updateShortMessage#914fbf11 flags:# out:flags.1?true mentioned:flags.4?true media_unread:flags.5?true silent:fl
updateShortChatMessage#16812688 flags:# out:flags.1?true mentioned:flags.4?true media_unread:flags.5?true silen
updateShort#78d4dec1 update:Update date:int = Updates;
updatesCombined#725b04c3 updates:Vector<Update> users:Vector<User> chats:Vector<Chat> date:int seq_start:int se
updates#74ae4240 updates:Vector<Update> users:Vector<User> chats:Vector<Chat> date:int seq:int = Updates;
updateShortSentMessage#11f1331c flags:# out:flags.1?true id:int pts:int pts_count:int date:int media:flags.9?Me

photos.photos#8dca6aa5 photos:Vector<Photo> users:Vector<User> = photos.Photos;
photos.photosSlice#15051f54 count:int photos:Vector<Photo> users:Vector<User> = photos.Photos;

photos.photo#20212ca8 photo:Photo users:Vector<User> = photos.Photo;

upload.file#96a18d5 type:storage.FileType mtime:int bytes:bytes = upload.File;
upload.fileCdnRedirect#f18cda44 dc_id:int file_token:bytes encryption_key:bytes encryption_iv:bytes file_hashes

dcOption#18b7a10d flags:# ipv6:flags.0?true media_only:flags.1?true tcpo_only:flags.2?true cdn:flags.3?true sta

config#330b4067 flags:# phonecalls_enabled:flags.1?true default_p2p_contacts:flags.3?true preload_featured_stic

nearestDc#8e1a1775 country:string this_dc:int nearest_dc:int = NearestDc;

help.appUpdate#1da7158f flags:# can_not_skip:flags.0?true id:int version:string text:string entities:Vector<Mes
help.noAppUpdate#c45a6536 = help.AppUpdate;

help.inviteText#18cb9f78 message:string = help.InviteText;

encryptedChatEmpty#ab7ec0a0 id:int = EncryptedChat;
encryptedChatWaiting#3bf703dc id:int access_hash:long date:int admin_id:int participant_id:int = EncryptedChat;
encryptedChatRequested#c878527e id:int access_hash:long date:int admin_id:int participant_id:int g_a:bytes = En
encryptedChat#fa56ce36 id:int access_hash:long date:int admin_id:int participant_id:int g_a_or_b:bytes key_fing
encryptedChatDiscarded#13d6dd27 id:int = EncryptedChat;

inputEncryptedChat#f141b5e1 chat_id:int access_hash:long = InputEncryptedChat;

encryptedFileEmpty#c21f497e = EncryptedFile;
encryptedFile#4a70994c id:long access_hash:long size:int dc_id:int key_fingerprint:int = EncryptedFile;

inputEncryptedFileEmpty#1837c364 = InputEncryptedFile;
inputEncryptedFileUploaded#64bd0306 id:long parts:int md5_checksum:string key_fingerprint:int = InputEncryptedF
inputEncryptedFile#5a17b5e5 id:long access_hash:long = InputEncryptedFile;
inputEncryptedFileBigUploaded#2dc173c8 id:long parts:int key_fingerprint:int = InputEncryptedFile;

encryptedMessage#ed18c118 random_id:long chat_id:int date:int bytes:bytes file:EncryptedFile = EncryptedMessage
encryptedMessageService#23734b06 random_id:long chat_id:int date:int bytes:bytes = EncryptedMessage;

messages.dhConfigNotModified#c0e24635 random:bytes = messages.DhConfig;
messages.dhConfig#2c221edd g:int p:bytes version:int random:bytes = messages.DhConfig;

messages.sentEncryptedMessage#560f8935 date:int = messages.SentEncryptedMessage;
messages.sentEncryptedFile#9493ff32 date:int file:EncryptedFile = messages.SentEncryptedMessage;

inputDocumentEmpty#72f0eaae = InputDocument;
inputDocument#1abfb575 id:long access_hash:long file_reference:bytes = InputDocument;

documentEmpty#36f8c871 id:long = Document;
document#9ba29cc1 flags:# id:long access_hash:long file_reference:bytes date:int mime_type:string size:int thum

help.support#17c6b5f6 phone_number:string user:User = help.Support;

notifyPeer#9fd40bd8 peer:Peer = NotifyPeer;
notifyUsers#b4c83b4c = NotifyPeer;
notifyChats#c007cec3 = NotifyPeer;
notifyBroadcasts#d612e8ef = NotifyPeer;

sendMessageTypingAction#16bf744e = SendMessageAction;
sendMessageCancelAction#fd5ec8f5 = SendMessageAction;
sendMessageRecordVideoAction#a187d66f = SendMessageAction;
sendMessageUploadVideoAction#e9763aec progress:int = SendMessageAction;
sendMessageRecordAudioAction#d52f73f7 = SendMessageAction;
sendMessageUploadAudioAction#f351d7ab progress:int = SendMessageAction;
sendMessageUploadPhotoAction#d1d34a26 progress:int = SendMessageAction;
sendMessageUploadDocumentAction#aa0cd9e4 progress:int = SendMessageAction;
sendMessageGeoLocationAction#176f8ba1 = SendMessageAction;
sendMessageChooseContactAction#628cbc6f = SendMessageAction;
sendMessageGamePlayAction#dd6a8f48 = SendMessageAction;
sendMessageRecordRoundAction#88f27fbc = SendMessageAction;
sendMessageUploadRoundAction#243e1c66 progress:int = SendMessageAction;

contacts.found#b3134d9d my_results:Vector<Peer> results:Vector<Peer> chats:Vector<Chat> users:Vector<User> = co

inputPrivacyKeyStatusTimestamp#4f96cb18 = InputPrivacyKey;
inputPrivacyKeyChatInvite#bdfb0426 = InputPrivacyKey;
inputPrivacyKeyPhoneCall#fabadc5f = InputPrivacyKey;
inputPrivacyKeyPhoneP2P#db9e70d2 = InputPrivacyKey;
inputPrivacyKeyForwards#a4dd4c08 = InputPrivacyKey;
inputPrivacyKeyProfilePhoto#5719bacc = InputPrivacyKey;
inputPrivacyKeyPhoneNumber#352dafa = InputPrivacyKey;
inputPrivacyKeyAddedByPhone#d1219bdd = InputPrivacyKey;

privacyKeyStatusTimestamp#bc2eab30 = PrivacyKey;
privacyKeyChatInvite#500e6dfa = PrivacyKey;
privacyKeyPhoneCall#3d662b7b = PrivacyKey;
privacyKeyPhoneP2P#39491cc8 = PrivacyKey;
privacyKeyForwards#69ec56a3 = PrivacyKey;
privacyKeyProfilePhoto#96151fed = PrivacyKey;
privacyKeyPhoneNumber#d19ae46d = PrivacyKey;
privacyKeyAddedByPhone#42ffd42b = PrivacyKey;

inputPrivacyValueAllowContacts#d09e07b = InputPrivacyRule;
inputPrivacyValueAllowAll#184b35ce = InputPrivacyRule;
inputPrivacyValueAllowUsers#131cc67f users:Vector<InputUser> = InputPrivacyRule;
inputPrivacyValueDisallowContacts#ba52007 = InputPrivacyRule;
inputPrivacyValueDisallowAll#d66b66c9 = InputPrivacyRule;
inputPrivacyValueDisallowUsers#90110467 users:Vector<InputUser> = InputPrivacyRule;
inputPrivacyValueAllowChatParticipants#4c81c1ba chats:Vector<int> = InputPrivacyRule;
inputPrivacyValueDisallowChatParticipants#d82363af chats:Vector<int> = InputPrivacyRule;

privacyValueAllowContacts#fffe1bac = PrivacyRule;
privacyValueAllowAll#65427b82 = PrivacyRule;
privacyValueAllowUsers#4d5bbe0c users:Vector<int> = PrivacyRule;
privacyValueDisallowContacts#f888fa1a = PrivacyRule;
privacyValueDisallowAll#8b73e763 = PrivacyRule;
privacyValueDisallowUsers#c7f49b7 users:Vector<int> = PrivacyRule;
privacyValueAllowChatParticipants#18be796b chats:Vector<int> = PrivacyRule;
privacyValueDisallowChatParticipants#acae0690 chats:Vector<int> = PrivacyRule;

account.privacyRules#50a04e45 rules:Vector<PrivacyRule> chats:Vector<Chat> users:Vector<User> = account.Privacy

accountDaysTTL#b8d0afdf days:int = AccountDaysTTL;

documentAttributeImageSize#6c37c15c w:int h:int = DocumentAttribute;
documentAttributeAnimated#11b58939 = DocumentAttribute;
documentAttributeSticker#6319d612 flags:# mask:flags.1?true alt:string stickerset:InputStickerSet mask_coords:f
documentAttributeVideo#ef02ce6 flags:# round_message:flags.0?true supports_streaming:flags.1?true duration:int
documentAttributeAudio#9852f9c6 flags:# voice:flags.10?true duration:int title:flags.0?string performer:flags.1
documentAttributeFilename#15590068 file_name:string = DocumentAttribute;
documentAttributeHasStickers#9801d2f7 = DocumentAttribute;

messages.stickersNotModified#f1749a22 = messages.Stickers;
messages.stickers#e4599bbd hash:int stickers:Vector<Document> = messages.Stickers;

stickerPack#12b299d4 emoticon:string documents:Vector<long> = StickerPack;

messages.allStickersNotModified#e86602c3 = messages.AllStickers;
messages.allStickers#edfd405f hash:int sets:Vector<StickerSet> = messages.AllStickers;

messages.affectedMessages#84d19185 pts:int pts_count:int = messages.AffectedMessages;

webPageEmpty#eb1477e8 id:long = WebPage;
webPagePending#c586da1c id:long date:int = WebPage;
webPage#fa64e172 flags:# id:long url:string display_url:string hash:int type:flags.0?string site_name:flags.1?s
webPageNotModified#85849473 = WebPage;

authorization#ad01d61d flags:# current:flags.0?true official_app:flags.1?true password_pending:flags.2?true has

account.authorizations#1250abde authorizations:Vector<Authorization> = account.Authorizations;

account.password#ad2641f8 flags:# has_recovery:flags.0?true has_secure_values:flags.1?true has_password:flags.2

account.passwordSettings#9a5c33e5 flags:# email:flags.0?string secure_settings:flags.1?SecureSecretSettings = a

account.passwordInputSettings#c23727c9 flags:# new_algo:flags.0?PasswordKdfAlgo new_password_hash:flags.0?bytes

auth.passwordRecovery#137948a5 email_pattern:string = auth.PasswordRecovery;

receivedNotifyMessage#a384b779 id:int flags:int = ReceivedNotifyMessage;

chatInviteEmpty#69df3769 = ExportedChatInvite;
chatInviteExported#fc2e05bc link:string = ExportedChatInvite;

chatInviteAlready#5a686d7c chat:Chat = ChatInvite;
chatInvite#dfc2f58e flags:# channel:flags.0?true broadcast:flags.1?true public:flags.2?true megagroup:flags.3?t

inputStickerSetEmpty#ffb62b95 = InputStickerSet;
inputStickerSetID#9de7a269 id:long access_hash:long = InputStickerSet;
inputStickerSetShortName#861cc8a0 short_name:string = InputStickerSet;
inputStickerSetAnimatedEmoji#28703c8 = InputStickerSet;

stickerSet#eeb46f27 flags:# archived:flags.1?true official:flags.2?true masks:flags.3?true animated:flags.5?tru

messages.stickerSet#b60a24a6 set:StickerSet packs:Vector<StickerPack> documents:Vector<Document> = messages.Sti

botCommand#c27ac8c7 command:string description:string = BotCommand;

botInfo#98e81d3a user_id:int description:string commands:Vector<BotCommand> = BotInfo;

keyboardButton#a2fa4880 text:string = KeyboardButton;
keyboardButtonUrl#258aff05 text:string url:string = KeyboardButton;
```

```
keyboardButtonUrl#258aff05 text:string url:string = KeyboardButton;
keyboardButtonCallback#683a5e46 text:string data:bytes = KeyboardButton;
keyboardButtonRequestPhone#b16a6c29 text:string = KeyboardButton;
keyboardButtonRequestGeoLocation#fc796b3f text:string = KeyboardButton;
keyboardButtonSwitchInline#568a748 flags:# same_peer:flags.0?true text:string query:string = KeyboardButton;
keyboardButtonGame#50f41ccf text:string = KeyboardButton;
keyboardButtonBuy#afd93fbb text:string = KeyboardButton;
keyboardButtonUrlAuth#10b78d29 flags:# text:string fwd_text:flags.0?string url:string button_id:int = KeyboardB
inputKeyboardButtonUrlAuth#d02e7fd4 flags:# request_write_access:flags.0?true text:string fwd_text:flags.1?stri

keyboardButtonRow#77608b83 buttons:Vector<KeyboardButton> = KeyboardButtonRow;

replyKeyboardHide#a03e5b85 flags:# selective:flags.2?true = ReplyMarkup;
replyKeyboardForceReply#f4108aa0 flags:# single_use:flags.1?true selective:flags.2?true = ReplyMarkup;
replyKeyboardMarkup#3502758c flags:# resize:flags.0?true single_use:flags.1?true selective:flags.2?true rows:Ve
replyInlineMarkup#48a30254 rows:Vector<KeyboardButtonRow> = ReplyMarkup;

messageEntityUnknown#bb92ba95 offset:int length:int = MessageEntity;
messageEntityMention#fa04579d offset:int length:int = MessageEntity;
messageEntityHashtag#6f635b0d offset:int length:int = MessageEntity;
messageEntityBotCommand#6cef8ac7 offset:int length:int = MessageEntity;
messageEntityUrl#6ed02538 offset:int length:int = MessageEntity;
messageEntityEmail#64e475c2 offset:int length:int = MessageEntity;
messageEntityBold#bd610bc9 offset:int length:int = MessageEntity;
messageEntityItalic#826f8b60 offset:int length:int = MessageEntity;
messageEntityCode#28a20571 offset:int length:int = MessageEntity;
messageEntityPre#73924be0 offset:int length:int language:string = MessageEntity;
messageEntityTextUrl#76a6d327 offset:int length:int url:string = MessageEntity;
messageEntityMentionName#352dca58 offset:int length:int user_id:int = MessageEntity;
inputMessageEntityMentionName#208e68c9 offset:int length:int user_id:InputUser = MessageEntity;
messageEntityPhone#9b69e34b offset:int length:int = MessageEntity;
messageEntityCashtag#4c4e743f offset:int length:int = MessageEntity;
messageEntityUnderline#9c4e7e8b offset:int length:int = MessageEntity;
messageEntityStrike#bf0693d4 offset:int length:int = MessageEntity;
messageEntityBlockquote#20df5d0 offset:int length:int = MessageEntity;

inputChannelEmpty#ee8c1e86 = InputChannel;
inputChannel#afeb712e channel_id:int access_hash:long = InputChannel;
inputChannelFromMessage#2a286531 peer:InputPeer msg_id:int channel_id:int = InputChannel;

contacts.resolvedPeer#7f077ad9 peer:Peer chats:Vector<Chat> users:Vector<User> = contacts.ResolvedPeer;

messageRange#ae30253 min_id:int max_id:int = MessageRange;

updates.channelDifferenceEmpty#3e11affb flags:# final:flags.0?true pts:int timeout:flags.1?int = updates.Channe
updates.channelDifferenceTooLong#a4bcc6fe flags:# final:flags.0?true timeout:flags.1?int dialog:Dialog messages
updates.channelDifference#2064674e flags:# final:flags.0?true pts:int timeout:flags.1?int new_messages:Vector<M

channelMessagesFilterEmpty#94d42ee7 = ChannelMessagesFilter;
channelMessagesFilter#cd77d957 flags:# exclude_new_messages:flags.1?true ranges:Vector<MessageRange> = ChannelM

channelParticipant#15ebac1d user_id:int date:int = ChannelParticipant;
channelParticipantSelf#a3289a6d user_id:int inviter_id:int date:int = ChannelParticipant;
channelParticipantCreator#808d15a4 flags:# user_id:int rank:flags.0?string = ChannelParticipant;
channelParticipantAdmin#ccbebbaf flags:# can_edit:flags.0?true self:flags.1?true user_id:int inviter_id:flags.1
channelParticipantBanned#1c0facaf flags:# left:flags.0?true user_id:int kicked_by:int date:int banned_rights:Ch

channelParticipantsRecent#de3f3c79 = ChannelParticipantsFilter;
channelParticipantsAdmins#b4608969 = ChannelParticipantsFilter;
channelParticipantsKicked#a3b54985 q:string = ChannelParticipantsFilter;
channelParticipantsBots#b0d1865b = ChannelParticipantsFilter;
channelParticipantsBanned#1427a5e1 q:string = ChannelParticipantsFilter;
channelParticipantsSearch#656ac4b q:string = ChannelParticipantsFilter;
channelParticipantsContacts#bb6ae88d q:string = ChannelParticipantsFilter;

channels.channelParticipants#f56ee2a8 count:int participants:Vector<ChannelParticipant> users:Vector<User> = ch
channels.channelParticipantsNotModified#f0173fe9 = channels.ChannelParticipants;

channels.channelParticipant#d0d9b163 participant:ChannelParticipant users:Vector<User> = channels.ChannelPartic

help.termsOfService#780a0310 flags:# popup:flags.0?true id:DataJSON text:string entities:Vector<MessageEntity>

foundGif#162ecc1f url:string thumb_url:string content_url:string content_type:string w:int h:int = FoundGif;
foundGifCached#9c750409 url:string photo:Photo document:Document = FoundGif;

messages.foundGifs#450a1c0a next_offset:int results:Vector<FoundGif> = messages.FoundGifs;

messages.savedGifsNotModified#e8025ca2 = messages.SavedGifs;
messages.savedGifs#2e0709a5 hash:int gifs:Vector<Document> = messages.SavedGifs;

inputBotInlineMessageMediaAuto#3380c786 flags:# message:string entities:flags.1?Vector<MessageEntity> reply_mar
inputBotInlineMessageText#3dcd7a87 flags:# no_webpage:flags.0?true message:string entities:flags.1?Vector<Messa
inputBotInlineMessageMediaGeo#c1b15d65 flags:# geo_point:InputGeoPoint period:int reply_markup:flags.2?ReplyMar
inputBotInlineMessageMediaVenue#417bbf11 flags:# geo_point:InputGeoPoint title:string address:string provider:s
inputBotInlineMessageMediaContact#a6edbffd flags:# phone_number:string first_name:string last_name:string vcard
inputBotInlineMessageGame#4b425864 flags:# reply_markup:flags.2?ReplyMarkup = InputBotInlineMessage;

inputBotInlineResult#88bf9319 flags:# id:string type:string title:flags.1?string description:flags.2?string url
inputBotInlineResultPhoto#a8d864a7 id:string type:string photo:InputPhoto send_message:InputBotInlineMessage =
inputBotInlineResultDocument#fff8fdc4 flags:# id:string type:string title:flags.1?string description:flags.2?st
inputBotInlineResultGame#4fa417f2 id:string short_name:string send_message:InputBotInlineMessage = InputBotInli

botInlineMessageMediaAuto#764cf810 flags:# message:string entities:flags.1?Vector<MessageEntity> reply_markup:f
botInlineMessageText#8c7f65e2 flags:# no_webpage:flags.0?true message:string entities:flags.1?Vector<MessageEnt
botInlineMessageMediaGeo#b722de65 flags:# geo:GeoPoint period:int reply_markup:flags.2?ReplyMarkup = BotInlineM
botInlineMessageMediaVenue#8a86659c flags:# geo:GeoPoint title:string address:string provider:string venue_id:s
botInlineMessageMediaContact#18d1cdc2 flags:# phone_number:string first_name:string last_name:string vcard:stri

botInlineResult#11965f3a flags:# id:string type:string title:flags.1?string description:flags.2?string url:flag
botInlineMediaResult#17db940b flags:# id:string type:string photo:flags.0?Photo document:flags.1?Document title

messages.botResults#947ca848 flags:# gallery:flags.0?true query_id:long next_offset:flags.1?string switch_pm:fl

exportedMessageLink#5dab1af4 link:string html:string = ExportedMessageLink;

messageFwdHeader#ec338270 flags:# from_id:flags.0?int from_name:flags.5?string date:int channel_id:flags.1?int

auth.codeTypeSms#72a3158c = auth.CodeType;
auth.codeTypeCall#741cd3e3 = auth.CodeType;
auth.codeTypeFlashCall#226ccefb = auth.CodeType;

auth.sentCodeTypeApp#3dbb5986 length:int = auth.SentCodeType;
auth.sentCodeTypeSms#c000bba2 length:int = auth.SentCodeType;
auth.sentCodeTypeCall#5353e5a7 length:int = auth.SentCodeType;
auth.sentCodeTypeFlashCall#ab03c6d9 pattern:string = auth.SentCodeType;

messages.botCallbackAnswer#36585ea4 flags:# alert:flags.1?true has_url:flags.3?true native_ui:flags.4?true mess

messages.messageEditData#26b5dde6 flags:# caption:flags.0?true = messages.MessageEditData;

inputBotInlineMessageID#890c3d89 dc_id:int id:long access_hash:long = InputBotInlineMessageID;

inlineBotSwitchPM#3c20629f text:string start_param:string = InlineBotSwitchPM;

messages.peerDialogs#3371c354 dialogs:Vector<Dialog> messages:Vector<Message> chats:Vector<Chat> users:Vector<U

topPeer#edcdc05b peer:Peer rating:double = TopPeer;

topPeerCategoryBotsPM#ab661b5b = TopPeerCategory;
topPeerCategoryBotsInline#148677e2 = TopPeerCategory;
topPeerCategoryCorrespondents#637b7ed = TopPeerCategory;
topPeerCategoryGroups#bd17a14a = TopPeerCategory;
```

```
topPeerCategoryChannels#161d9628 = TopPeerCategory;
topPeerCategoryPhoneCalls#1e76a78c = TopPeerCategory;
topPeerCategoryForwardUsers#a8406ca9 = TopPeerCategory;
topPeerCategoryForwardChats#fbeec0f0 = TopPeerCategory;

topPeerCategoryPeers#fb834291 category:TopPeerCategory count:int peers:Vector<TopPeer> = TopPeerCategoryPeers;

contacts.topPeersNotModified#de266ef5 = contacts.TopPeers;
contacts.topPeers#70b772a8 categories:Vector<TopPeerCategoryPeers> chats:Vector<Chat> users:Vector<User> = cont
contacts.topPeersDisabled#b52c939d = contacts.TopPeers;

draftMessageEmpty#1b0c841a flags:# date:flags.0?int = DraftMessage;
draftMessage#fd8e711f flags:# no_webpage:flags.1?true reply_to_msg_id:flags.0?int message:string entities:flags

messages.featuredStickersNotModified#4ede3cf = messages.FeaturedStickers;
messages.featuredStickers#f89d88e5 hash:int sets:Vector<StickerSetCovered> unread:Vector<long> = messages.Featu

messages.recentStickersNotModified#b17f890 = messages.RecentStickers;
messages.recentStickers#22f3afb3 hash:int packs:Vector<StickerPack> stickers:Vector<Document> dates:Vector<int>

messages.archivedStickers#4fcba9c8 count:int sets:Vector<StickerSetCovered> = messages.ArchivedStickers;

messages.stickerSetInstallResultSuccess#38641628 = messages.StickerSetInstallResult;
messages.stickerSetInstallResultArchive#35e410a8 sets:Vector<StickerSetCovered> = messages.StickerSetInstallRes

stickerSetCovered#6410a5d2 set:StickerSet cover:Document = StickerSetCovered;
stickerSetMultiCovered#3407e51b set:StickerSet covers:Vector<Document> = StickerSetCovered;

maskCoords#aed6dbb2 n:int x:double y:double zoom:double = MaskCoords;

inputStickeredMediaPhoto#4a992157 id:InputPhoto = InputStickeredMedia;
inputStickeredMediaDocument#438865b id:InputDocument = InputStickeredMedia;

game#bdf9653b flags:# id:long access_hash:long short_name:string title:string description:string photo:Photo do

inputGameID#32c3e77 id:long access_hash:long = InputGame;
inputGameShortName#c331e80a bot_id:InputUser short_name:string = InputGame;

highScore#58fffcd0 pos:int user_id:int score:int = HighScore;

messages.highScores#9a3bfd99 scores:Vector<HighScore> users:Vector<User> = messages.HighScores;

textEmpty#dc3d824f = RichText;
textPlain#744694e0 text:string = RichText;
textBold#6724abc4 text:RichText = RichText;
textItalic#d912a59c text:RichText = RichText;
textUnderline#c12622c4 text:RichText = RichText;
textStrike#9bf8bb95 text:RichText = RichText;
textFixed#6c3f19b9 text:RichText = RichText;
textUrl#3c2884c1 text:RichText url:string webpage_id:long = RichText;
textEmail#de5a0dd6 text:RichText email:string = RichText;
textConcat#7e6260d7 texts:Vector<RichText> = RichText;
textSubscript#ed6a8504 text:RichText = RichText;
textSuperscript#c7fb5e01 text:RichText = RichText;
textMarked#34b8621 text:RichText = RichText;
textPhone#1ccb966a text:RichText phone:string = RichText;
textImage#81ccf4f document_id:long w:int h:int = RichText;
textAnchor#35553762 text:RichText name:string = RichText;

pageBlockUnsupported#13567e8a = PageBlock;
pageBlockTitle#70abc3fd text:RichText = PageBlock;
pageBlockSubtitle#8ffa9a1f text:RichText = PageBlock;
pageBlockAuthorDate#baafe5e0 author:RichText published_date:int = PageBlock;
pageBlockHeader#bfd064ec text:RichText = PageBlock;
pageBlockSubheader#f12bb6e1 text:RichText = PageBlock;
pageBlockParagraph#467a0766 text:RichText = PageBlock;
pageBlockPreformatted#c070d93e text:RichText language:string = PageBlock;
pageBlockFooter#48870999 text:RichText = PageBlock;
pageBlockDivider#db20b188 = PageBlock;
pageBlockAnchor#ce0d37b0 name:string = PageBlock;
pageBlockList#e4e88011 items:Vector<PageListItem> = PageBlock;
pageBlockBlockquote#263d7c26 text:RichText caption:RichText = PageBlock;
pageBlockPullquote#4f4456d3 text:RichText caption:RichText = PageBlock;
pageBlockPhoto#1759c560 flags:# photo_id:long caption:PageCaption url:flags.0?string webpage_id:flags.0?long =
pageBlockVideo#7c8fe7b6 flags:# autoplay:flags.0?true loop:flags.1?true video_id:long caption:PageCaption = Pag
pageBlockCover#39f23300 cover:PageBlock = PageBlock;
pageBlockEmbed#a8718dc5 flags:# full_width:flags.0?true allow_scrolling:flags.3?true url:flags.1?string html:fl
pageBlockEmbedPost#f259a80b url:string webpage_id:long author_photo_id:long author:string date:int blocks:Vecto
pageBlockCollage#65a0fa4d items:Vector<PageBlock> caption:PageCaption = PageBlock;
pageBlockSlideshow#31f9590 items:Vector<PageBlock> caption:PageCaption = PageBlock;
pageBlockChannel#ef1751b5 channel:Chat = PageBlock;
pageBlockAudio#804361ea audio_id:long caption:PageCaption = PageBlock;
pageBlockKicker#1e148390 text:RichText = PageBlock;
pageBlockTable#bf4dea82 flags:# bordered:flags.0?true striped:flags.1?true title:RichText rows:Vector<PageTable
pageBlockOrderedList#9a8ae1e1 items:Vector<PageListOrderedItem> = PageBlock;
pageBlockDetails#76768bed flags:# open:flags.0?true blocks:Vector<PageBlock> title:RichText = PageBlock;
pageBlockRelatedArticles#16115a96 title:RichText articles:Vector<PageRelatedArticle> = PageBlock;
pageBlockMap#a44f3ef6 geo:GeoPoint zoom:int w:int h:int caption:PageCaption = PageBlock;

phoneCallDiscardReasonMissed#85e42301 = PhoneCallDiscardReason;
phoneCallDiscardReasonDisconnect#e095c1a0 = PhoneCallDiscardReason;
phoneCallDiscardReasonHangup#57adc690 = PhoneCallDiscardReason;
phoneCallDiscardReasonBusy#faf7e8c9 = PhoneCallDiscardReason;

dataJSON#7d748d04 data:string = DataJSON;

labeledPrice#cb296bf8 label:string amount:long = LabeledPrice;

invoice#c30aa358 flags:# test:flags.0?true name_requested:flags.1?true phone_requested:flags.2?true email_reque

paymentCharge#ea02c27e id:string provider_charge_id:string = PaymentCharge;

postAddress#1e8caaeb street_line1:string street_line2:string city:string state:string country_iso2:string post_

paymentRequestedInfo#909c3f94 flags:# name:flags.0?string phone:flags.1?string email:flags.2?string shipping_ad

paymentSavedCredentialsCard#cdc27a1f id:string title:string = PaymentSavedCredentials;

webDocument#1c570ed1 url:string access_hash:long size:int mime_type:string attributes:Vector<DocumentAttribute>
webDocumentNoProxy#f9c8bcc6 url:string size:int mime_type:string attributes:Vector<DocumentAttribute> = WebDocu

inputWebDocument#9bed434d url:string size:int mime_type:string attributes:Vector<DocumentAttribute> = InputWebD

inputWebFileLocation#c239d686 url:string access_hash:long = InputWebFileLocation;
inputWebFileGeoPointLocation#9f2221c9 geo_point:InputGeoPoint access_hash:long w:int h:int zoom:int scale:int =

upload.webFile#21e753bc size:int mime_type:string file_type:storage.FileType mtime:int bytes:bytes = upload.Web

payments.paymentForm#3f56aea3 flags:# can_save_credentials:flags.2?true password_missing:flags.3?true bot_id:in

payments.validatedRequestedInfo#d1451883 flags:# id:flags.0?string shipping_options:flags.1?Vector<ShippingOpti

payments.paymentResult#4e5f810d updates:Updates = payments.PaymentResult;
payments.paymentVerificationNeeded#d8411139 url:string = payments.PaymentResult;

payments.paymentReceipt#500911e1 flags:# date:int bot_id:int invoice:Invoice provider_id:int info:flags.0?Payme

payments.savedInfo#fb8fe43c flags:# has_saved_credentials:flags.1?true saved_info:flags.0?PaymentRequestedInfo

inputPaymentCredentialsSaved#c10eb2cf id:string tmp_password:bytes = InputPaymentCredentials;
inputPaymentCredentials#3417d728 flags:# save:flags.0?true data:DataJSON = InputPaymentCredentials;
inputPaymentCredentialsApplePay#aa1c39f payment_data:DataJSON = InputPaymentCredentials;
```



```
inputPaymentCredentialsApplePay#aa1c39f payment_data:DataJSON = InputPaymentCredentials;
inputPaymentCredentialsAndroidPay#ca05d50e payment_token:DataJSON google_transaction_id:string = InputPaymentCr

account.tmpPassword#db64fd34 tmp_password:bytes valid_until:int = account.TmpPassword;

shippingOption#b6213cdf id:string title:string prices:Vector<LabeledPrice> = ShippingOption;

inputStickerSetItem#ffa0a496 flags:# document:InputDocument emoji:string mask_coords:flags.0?MaskCoords = Input

inputPhoneCall#1e36fded id:long access_hash:long = InputPhoneCall;

phoneCallEmpty#5366c915 id:long = PhoneCall;
phoneCallWaiting#1b8f4ad1 flags:# video:flags.5?true id:long access_hash:long date:int admin_id:int participant
phoneCallRequested#87eabb53 flags:# video:flags.5?true id:long access_hash:long date:int admin_id:int participa
phoneCallAccepted#997c454a flags:# video:flags.5?true id:long access_hash:long date:int admin_id:int participan
phoneCall#8742ae7f flags:# p2p_allowed:flags.5?true id:long access_hash:long date:int admin_id:int participant_
phoneCallDiscarded#50ca4de1 flags:# need_rating:flags.2?true need_debug:flags.3?true video:flags.5?true id:long

phoneConnection#9d4c17c0 id:long ip:string ipv6:string port:int peer_tag:bytes = PhoneConnection;

phoneCallProtocol#a2bb35cb flags:# udp_p2p:flags.0?true udp_reflector:flags.1?true min_layer:int max_layer:int

phone.phoneCall#ec82e140 phone_call:PhoneCall users:Vector<User> = phone.PhoneCall;

upload.cdnFileReuploadNeeded#eea8e46e request_token:bytes = upload.CdnFile;
upload.cdnFile#a99fca4f bytes:bytes = upload.CdnFile;

cdnPublicKey#c982eaba dc_id:int public_key:string = CdnPublicKey;

cdnConfig#5725e40a public_keys:Vector<CdnPublicKey> = CdnConfig;

langPackString#cad181f6 key:string value:string = LangPackString;
langPackStringPluralized#6c47ac9f flags:# key:string zero_value:flags.0?string one_value:flags.1?string two_val
langPackStringDeleted#2979eeb2 key:string = LangPackString;

langPackDifference#f385c1f6 lang_code:string from_version:int version:int strings:Vector<LangPackString> = Lang

langPackLanguage#eeca5ce3 flags:# official:flags.0?true rtl:flags.2?true beta:flags.3?true name:string native_n

channelAdminLogEventActionChangeTitle#e6dfb825 prev_value:string new_value:string = ChannelAdminLogEventAction;
channelAdminLogEventActionChangeAbout#55188a2e prev_value:string new_value:string = ChannelAdminLogEventAction;
channelAdminLogEventActionChangeUsername#6a4afc38 prev_value:string new_value:string = ChannelAdminLogEventActi
channelAdminLogEventActionChangePhoto#434bd2af prev_photo:Photo new_photo:Photo = ChannelAdminLogEventAction;
channelAdminLogEventActionToggleInvites#1b7907ae new_value:Bool = ChannelAdminLogEventAction;
channelAdminLogEventActionToggleSignatures#26ae0971 new_value:Bool = ChannelAdminLogEventAction;
channelAdminLogEventActionUpdatePinned#e9e82c18 message:Message = ChannelAdminLogEventAction;
channelAdminLogEventActionEditMessage#709b2405 prev_message:Message new_message:Message = ChannelAdminLogEventA
channelAdminLogEventActionDeleteMessage#42e047bb message:Message = ChannelAdminLogEventAction;
channelAdminLogEventActionParticipantJoin#183040d3 = ChannelAdminLogEventAction;
channelAdminLogEventActionParticipantLeave#f89777f2 = ChannelAdminLogEventAction;
channelAdminLogEventActionParticipantInvite#e31c34d8 participant:ChannelParticipant = ChannelAdminLogEventActio
channelAdminLogEventActionParticipantToggleBan#e6d83d7e prev_participant:ChannelParticipant new_participant:Cha
channelAdminLogEventActionParticipantToggleAdmin#d5676710 prev_participant:ChannelParticipant new_participant:C
channelAdminLogEventActionChangeStickerSet#b1c3caa7 prev_stickerset:InputStickerSet new_stickerset:InputSticker
channelAdminLogEventActionTogglePreHistoryHidden#5f5c95f1 new_value:Bool = ChannelAdminLogEventAction;
channelAdminLogEventActionDefaultBannedRights#2df5fc0a prev_banned_rights:ChatBannedRights new_banned_rights:Ch
channelAdminLogEventActionStopPoll#8f079643 message:Message = ChannelAdminLogEventAction;
channelAdminLogEventActionChangeLinkedChat#a26f881b prev_value:int new_value:int = ChannelAdminLogEventAction;
channelAdminLogEventActionChangeLocation#e6b76ae prev_value:ChannelLocation new_value:ChannelLocation = Channel
channelAdminLogEventActionToggleSlowMode#53909779 prev_value:int new_value:int = ChannelAdminLogEventAction;

channelAdminLogEvent#3b5a3e40 id:long date:int user_id:int action:ChannelAdminLogEventAction = ChannelAdminLogE

channels.adminLogResults#ed8af74d events:Vector<ChannelAdminLogEvent> chats:Vector<Chat> users:Vector<User> = c

channelAdminLogEventsFilter#ea107ae4 flags:# join:flags.0?true leave:flags.1?true invite:flags.2?true ban:flags

popularContact#5ce14175 client_id:long importers:int = PopularContact;

messages.favedStickersNotModified#9e8fa6d3 = messages.FavedStickers;
messages.favedStickers#f37f2f16 hash:int packs:Vector<StickerPack> stickers:Vector<Document> = messages.FavedSt

recentMeUrlUnknown#46e1d13d url:string = RecentMeUrl;
recentMeUrlUser#8dbc3336 url:string user_id:int = RecentMeUrl;
recentMeUrlChat#a01b22f9 url:string chat_id:int = RecentMeUrl;
recentMeUrlChatInvite#eb49081d url:string chat_invite:ChatInvite = RecentMeUrl;
recentMeUrlStickerSet#bc0a57dc url:string set:StickerSetCovered = RecentMeUrl;

help.recentMeUrls#e0310d7 urls:Vector<RecentMeUrl> chats:Vector<Chat> users:Vector<User> = help.RecentMeUrls;

inputSingleMedia#1cc6e91f flags:# media:InputMedia random_id:long message:string entities:flags.0?Vector<Messag

webAuthorization#cac943f2 hash:long bot_id:int domain:string browser:string platform:string date_created:int da

account.webAuthorizations#ed56c9fc authorizations:Vector<WebAuthorization> users:Vector<User> = account.WebAuth

inputMessageID#a676a322 id:int = InputMessage;
inputMessageReplyTo#bad88395 id:int = InputMessage;
inputMessagePinned#86872538 = InputMessage;

inputDialogPeer#fcaafeb7 peer:InputPeer = InputDialogPeer;
inputDialogPeerFolder#64600527 folder_id:int = InputDialogPeer;

dialogPeer#e56dbf05 peer:Peer = DialogPeer;
dialogPeerFolder#514519e2 folder_id:int = DialogPeer;

messages.foundStickerSetsNotModified#d54b65d = messages.FoundStickerSets;
messages.foundStickerSets#5108d648 hash:int sets:Vector<StickerSetCovered> = messages.FoundStickerSets;

fileHash#6242c773 offset:int limit:int hash:bytes = FileHash;

inputClientProxy#75588b3f address:string port:int = InputClientProxy;

help.proxyDataEmpty#e09e1fb8 expires:int = help.ProxyData;
help.proxyDataPromo#2bf7ee23 expires:int peer:Peer chats:Vector<Chat> users:Vector<User> = help.ProxyData;

help.termsOfServiceUpdateEmpty#e3309f7f expires:int = help.TermsOfServiceUpdate;
help.termsOfServiceUpdate#28ecf961 expires:int terms_of_service:help.TermsOfService = help.TermsOfServiceUpdate

inputSecureFileUploaded#3334b0f0 id:long parts:int md5_checksum:string file_hash:bytes secret:bytes = InputSecu
inputSecureFile#5367e5be id:long access_hash:long = InputSecureFile;

secureFileEmpty#64199744 = SecureFile;
secureFile#e0277a62 id:long access_hash:long size:int dc_id:int date:int file_hash:bytes secret:bytes = SecureF

secureData#8aeabec3 data:bytes data_hash:bytes secret:bytes = SecureData;

securePlainPhone#7d6099dd phone:string = SecurePlainData;
securePlainEmail#21ec5a5f email:string = SecurePlainData;

secureValueTypePersonalDetails#9d2a81e3 = SecureValueType;
secureValueTypePassport#3dac6a00 = SecureValueType;
secureValueTypeDriverLicense#6e425c4 = SecureValueType;
secureValueTypeIdentityCard#a0d0744b = SecureValueType;
secureValueTypeInternalPassport#99a48f23 = SecureValueType;
secureValueTypeAddress#cbe31e26 = SecureValueType;
secureValueTypeUtilityBill#fc36954e = SecureValueType;
secureValueTypeBankStatement#89137c0d = SecureValueType;
secureValueTypeRentalAgreement#8b883488 = SecureValueType;
secureValueTypePassportRegistration#99e3806a = SecureValueType;
secureValueTypeTemporaryRegistration#ea02ec33 = SecureValueType;
secureValueTypePhone#b320aadb = SecureValueType;
```

```
secureValueTypeEmail#8e3ca7ee = SecureValueType;
```

```
secureValue#187fa0ca flags:# type:SecureValueType data:flags.0?SecureData front_side:flags.1?SecureFile reverse

inputSecureValue#db21d0a7 flags:# type:SecureValueType data:flags.0?SecureData front_side:flags.1?InputSecureFi

secureValueHash#ed1ecdb0 type:SecureValueType hash:bytes = SecureValueHash;

secureValueErrorData#e8a40bd9 type:SecureValueType data_hash:bytes field:string text:string = SecureValueError;
secureValueErrorFrontSide#be3dfa type:SecureValueType file_hash:bytes text:string = SecureValueError;
secureValueErrorReverseSide#868a2aa5 type:SecureValueType file_hash:bytes text:string = SecureValueError;
secureValueErrorSelfie#e537ced6 type:SecureValueType file_hash:bytes text:string = SecureValueError;
secureValueErrorFile#7a700873 type:SecureValueType file_hash:bytes text:string = SecureValueError;
secureValueErrorFiles#666220e9 type:SecureValueType file_hash:Vector<bytes> text:string = SecureValueError;
secureValueError#869d758f type:SecureValueType hash:bytes text:string = SecureValueError;
secureValueErrorTranslationFile#a1144770 type:SecureValueType file_hash:bytes text:string = SecureValueError;
secureValueErrorTranslationFiles#34636dd8 type:SecureValueType file_hash:Vector<bytes> text:string = SecureValu

secureCredentialsEncrypted#33f0ea47 data:bytes hash:bytes secret:bytes = SecureCredentialsEncrypted;

account.authorizationForm#ad2e1cd8 flags:# required_types:Vector<SecureRequiredType> values:Vector<SecureValue>

account.sentEmailCode#811f854f email_pattern:string length:int = account.SentEmailCode;

help.deepLinkInfoEmpty#66afa166 = help.DeepLinkInfo;
help.deepLinkInfo#6a4ee832 flags:# update_app:flags.0?true message:string entities:flags.1?Vector<MessageEntity

savedPhoneContact#1142bd56 phone:string first_name:string last_name:string date:int = SavedContact;

account.takeout#4dba4501 id:long = account.Takeout;

passwordKdfAlgoUnknown#d45ab096 = PasswordKdfAlgo;
passwordKdfAlgoSHA256SHA256PBKDF2HMACSHA512iter100000SHA256ModPow#3a912d4a salt1:bytes salt2:bytes g:int p:byte

securePasswordKdfAlgoUnknown#4a8537 = SecurePasswordKdfAlgo;
securePasswordKdfAlgoPBKDF2HMACSHA512iter100000#bbf2dda0 salt:bytes = SecurePasswordKdfAlgo;
securePasswordKdfAlgoSHA512#86471d92 salt:bytes = SecurePasswordKdfAlgo;

secureSecretSettings#1527bcac secure_algo:SecurePasswordKdfAlgo secure_secret:bytes secure_secret_id:long = Sec

inputCheckPasswordEmpty#9880f658 = InputCheckPasswordSRP;
inputCheckPasswordSRP#d27ff082 srp_id:long A:bytes M1:bytes = InputCheckPasswordSRP;

secureRequiredType#829d99da flags:# native_names:flags.0?true selfie_required:flags.1?true translation_required
secureRequiredTypeOneOf#27477b4 types:Vector<SecureRequiredType> = SecureRequiredType;

help.passportConfigNotModified#bfb9f457 = help.PassportConfig;
help.passportConfig#a098d6af hash:int countries_langs:DataJSON = help.PassportConfig;

inputAppEvent#1d1b1245 time:double type:string peer:long data:JSONValue = InputAppEvent;

jsonObjectValue#c0de1bd9 key:string value:JSONValue = JSONObjectValue;

jsonNull#3f6d7b68 = JSONValue;
jsonBool#c7345e6a value:Bool = JSONValue;
jsonNumber#2be0dfa4 value:double = JSONValue;
jsonString#b71e767a value:string = JSONValue;
jsonArray#f7444763 value:Vector<JSONValue> = JSONValue;
jsonObject#99c1d49d value:Vector<JSONObjectValue> = JSONValue;

pageTableCell#34566b6a flags:# header:flags.0?true align_center:flags.3?true align_right:flags.4?true valign_mi

pageTableRow#e0c0c5e5 cells:Vector<PageTableCell> = PageTableRow;

pageCaption#6f747657 text:RichText credit:RichText = PageCaption;

pageListItemText#b92fb6cd text:RichText = PageListItem;
pageListItemBlocks#25e073fc blocks:Vector<PageBlock> = PageListItem;

pageListOrderedItemText#5e068047 num:string text:RichText = PageListOrderedItem;
pageListOrderedItemBlocks#98dd8936 num:string blocks:Vector<PageBlock> = PageListOrderedItem;

pageRelatedArticle#b390dc08 flags:# url:string webpage_id:long title:flags.0?string description:flags.1?string

page#ae891bec flags:# part:flags.0?true rtl:flags.1?true v2:flags.2?true url:string blocks:Vector<PageBlock> ph

help.supportName#8c05f1c9 name:string = help.SupportName;

help.userInfoEmpty#f3ae2eed = help.UserInfo;
help.userInfo#1eb3758 message:string entities:Vector<MessageEntity> author:string date:int = help.UserInfo;

pollAnswer#6ca9c2e9 text:string option:bytes = PollAnswer;

poll#d5529d06 id:long flags:# closed:flags.0?true question:string answers:Vector<PollAnswer> = Poll;

pollAnswerVoters#3b6ddad2 flags:# chosen:flags.0?true option:bytes voters:int = PollAnswerVoters;

pollResults#5755785a flags:# min:flags.0?true results:flags.1?Vector<PollAnswerVoters> total_voters:flags.2?int

chatOnlines#f041e250 onlines:int = ChatOnlines;

statsURL#47a971e0 url:string = StatsURL;

chatAdminRights#5fb224d5 flags:# change_info:flags.0?true post_messages:flags.1?true edit_messages:flags.2?true

chatBannedRights#9f120418 flags:# view_messages:flags.0?true send_messages:flags.1?true send_media:flags.2?true

inputWallPaper#e630b979 id:long access_hash:long = InputWallPaper;
inputWallPaperSlug#72091c80 slug:string = InputWallPaper;

account.wallPapersNotModified#1c199183 = account.WallPapers;
account.wallPapers#702b65a9 hash:int wallpapers:Vector<WallPaper> = account.WallPapers;

codeSettings#debebe83 flags:# allow_flashcall:flags.0?true current_number:flags.1?true allow_app_hash:flags.4?t

wallPaperSettings#a12f40b8 flags:# blur:flags.1?true motion:flags.2?true background_color:flags.0?int intensity

autoDownloadSettings#d246fd47 flags:# disabled:flags.0?true video_preload_large:flags.1?true audio_preload_next

account.autoDownloadSettings#63cacf26 low:AutoDownloadSettings medium:AutoDownloadSettings high:AutoDownloadSet

emojiKeyword#d5b3b9f9 keyword:string emoticons:Vector<string> = EmojiKeyword;
emojiKeywordDeleted#236df622 keyword:string emoticons:Vector<string> = EmojiKeyword;

emojiKeywordsDifference#5cc761bd lang_code:string from_version:int version:int keywords:Vector<EmojiKeyword> =

emojiURL#a575739d url:string = EmojiURL;

emojiLanguage#b3fb5361 lang_code:string = EmojiLanguage;

fileLocationToBeDeprecated#bc7fc6cd volume_id:long local_id:int = FileLocation;

folder#ff544e65 flags:# autofill_new_broadcasts:flags.0?true autofill_public_groups:flags.1?true autofill_new_c

inputFolderPeer#fbd2c296 peer:InputPeer folder_id:int = InputFolderPeer;

folderPeer#e9baa668 peer:Peer folder_id:int = FolderPeer;

messages.searchCounter#e844ebff flags:# inexact:flags.1?true filter:MessagesFilter count:int = messages.SearchC

urlAuthResultRequest#92d33a0e flags:# request_write_access:flags.0?true bot:User domain:string = UrlAuthResult;
urlAuthResultAccepted#8f8c0e4e url:string = UrlAuthResult;
```

```
urlAuthResultDefault#a9d6db1f = UrlAuthResult;

channelLocationEmpty#bfb5ad8b = ChannelLocation;
channelLocation#209b82db geo_point:GeoPoint address:string = ChannelLocation;

peerLocated#ca461b5d peer:Peer expires:int distance:int = PeerLocated;

restrictionReason#d072acb4 platform:string reason:string text:string = RestrictionReason;

inputTheme#3c5693e9 id:long access_hash:long = InputTheme;
inputThemeSlug#f5890df1 slug:string = InputTheme;

themeDocumentNotModified#483d270c = Theme;
theme#f7d90ce0 flags:# creator:flags.0?true default:flags.1?true id:long access_hash:long slug:string title:str

account.themesNotModified#f41eb622 = account.Themes;
account.themes#7f676421 hash:int themes:Vector<Theme> = account.Themes;

---functions---

invokeAfterMsg#cb9f372d {X:Type} msg_id:long query:!X = X;
invokeAfterMsgs#3dc4b4f0 {X:Type} msg_ids:Vector<long> query:!X = X;
initConnection#785188b8 {X:Type} flags:# api_id:int device_model:string system_version:string app_version:strin
invokeWithLayer#da9b0d0d {X:Type} layer:int query:!X = X;
invokeWithoutUpdates#bf9459b7 {X:Type} query:!X = X;
invokeWithMessagesRange#365275f2 {X:Type} range:MessageRange query:!X = X;
invokeWithTakeout#aca9fd2e {X:Type} takeout_id:long query:!X = X;

auth.sendCode#a677244f phone_number:string api_id:int api_hash:string settings:CodeSettings = auth.SentCode;
auth.signUp#80eee427 phone_number:string phone_code_hash:string first_name:string last_name:string = auth.Autho
auth.signIn#bcd51581 phone_number:string phone_code_hash:string phone_code:string = auth.Authorization;
auth.logOut#5717da40 = Bool;
auth.resetAuthorizations#9fab0d1a = Bool;
auth.exportAuthorization#e5bfffcd dc_id:int = auth.ExportedAuthorization;
auth.importAuthorization#e3ef9613 id:int bytes:bytes = auth.Authorization;
auth.bindTempAuthKey#cdd42a05 perm_auth_key_id:long nonce:long expires_at:int encrypted_message:bytes = Bool;
auth.importBotAuthorization#67a3ff2c flags:int api_id:int api_hash:string bot_auth_token:string = auth.Authoriz
auth.checkPassword#d18b4d16 password:InputCheckPasswordSRP = auth.Authorization;
auth.requestPasswordRecovery#d897bc66 = auth.PasswordRecovery;
auth.recoverPassword#4ea56e92 code:string = auth.Authorization;
auth.resendCode#3ef1a9bf phone_number:string phone_code_hash:string = auth.SentCode;
auth.cancelCode#1f040578 phone_number:string phone_code_hash:string = Bool;
auth.dropTempAuthKeys#8e48a188 except_auth_keys:Vector<long> = Bool;

account.registerDevice#68976c6f flags:# no_muted:flags.0?true token_type:int token:string app_sandbox:Bool secr
account.unregisterDevice#3076c4bf token_type:int token:string other_uids:Vector<int> = Bool;
account.updateNotifySettings#84be5b93 peer:InputNotifyPeer settings:InputPeerNotifySettings = Bool;
account.getNotifySettings#12b3ad31 peer:InputNotifyPeer = PeerNotifySettings;
account.resetNotifySettings#db7e1747 = Bool;
account.updateProfile#78515775 flags:# first_name:flags.0?string last_name:flags.1?string about:flags.2?string
account.updateStatus#6628562c offline:Bool = Bool;
account.getWallPapers#aabb1763 hash:int = account.WallPapers;
account.reportPeer#ae189d5f peer:InputPeer reason:ReportReason = Bool;
account.checkUsername#2714d86c username:string = Bool;
account.updateUsername#3e0bdd7c username:string = User;
account.getPrivacy#dadbc950 key:InputPrivacyKey = account.PrivacyRules;
account.setPrivacy#c9f81ce8 key:InputPrivacyKey rules:Vector<InputPrivacyRule> = account.PrivacyRules;
account.deleteAccount#418d4e0b reason:string = Bool;
account.getAccountTTL#8fc711d = AccountDaysTTL;
account.setAccountTTL#2442485e ttl:AccountDaysTTL = Bool;
account.sendChangePhoneCode#82574ae5 phone_number:string settings:CodeSettings = auth.SentCode;
account.changePhone#70c32edb phone_number:string phone_code_hash:string phone_code:string = User;
account.updateDeviceLocked#38df3532 period:int = Bool;
account.getAuthorizations#e320c158 = account.Authorizations;
account.resetAuthorization#df77f3bc hash:long = Bool;
account.getPassword#548a30f5 = account.Password;
account.getPasswordSettings#9cd4eaf9 password:InputCheckPasswordSRP = account.PasswordSettings;
account.updatePasswordSettings#a59b102f password:InputCheckPasswordSRP new_settings:account.PasswordInputSettin
account.sendConfirmPhoneCode#1b3faa88 hash:string settings:CodeSettings = auth.SentCode;
account.confirmPhone#5f2178c3 phone_code_hash:string phone_code:string = Bool;
account.getTmpPassword#449e0b51 password:InputCheckPasswordSRP period:int = account.TmpPassword;
account.getWebAuthorizations#182e6d6f = account.WebAuthorizations;
account.resetWebAuthorization#2d01b9ef hash:long = Bool;
account.resetWebAuthorizations#682d2594 = Bool;
account.getAllSecureValues#b288bc7d = Vector<SecureValue>;
account.getSecureValue#73665bc2 types:Vector<SecureValueType> = Vector<SecureValue>;
account.saveSecureValue#899fe31d value:InputSecureValue secure_secret_id:long = SecureValue;
account.deleteSecureValue#b880bc4b types:Vector<SecureValueType> = Bool;
account.getAuthorizationForm#b86ba8e1 bot_id:int scope:string public_key:string = account.AuthorizationForm;
account.acceptAuthorization#e7027c94 bot_id:int scope:string public_key:string value_hashes:Vector<SecureValueH
account.sendVerifyPhoneCode#a5a356f9 phone_number:string settings:CodeSettings = auth.SentCode;
account.verifyPhone#4dd3a7f6 phone_number:string phone_code_hash:string phone_code:string = Bool;
account.sendVerifyEmailCode#7011509f email:string = account.SentEmailCode;
account.verifyEmail#ecba39db email:string code:string = Bool;
account.initTakeoutSession#f05b4804 flags:# contacts:flags.0?true message_users:flags.1?true message_chats:flag
account.finishTakeoutSession#1d2652ee flags:# success:flags.0?true = Bool;
account.confirmPasswordEmail#8fdf1920 code:string = Bool;
account.resendPasswordEmail#7a7f2a15 = Bool;
account.cancelPasswordEmail#c1cbd5b6 = Bool;
account.getContactSignUpNotification#9f07c728 = Bool;
account.setContactSignUpNotification#cff43f61 silent:Bool = Bool;
account.getNotifyExceptions#53577479 flags:# compare_sound:flags.1?true peer:flags.0?InputNotifyPeer = Updates;
account.getWallPaper#fc8ddbea wallpaper:InputWallPaper = WallPaper;
account.uploadWallPaper#dd853661 file:InputFile mime_type:string settings:WallPaperSettings = WallPaper;
account.saveWallPaper#6c5a5b37 wallpaper:InputWallPaper unsave:Bool settings:WallPaperSettings = Bool;
account.installWallPaper#feed5769 wallpaper:InputWallPaper settings:WallPaperSettings = Bool;
account.resetWallPapers#bb3b9804 = Bool;
account.getAutoDownloadSettings#56da0b3f = account.AutoDownloadSettings;
account.saveAutoDownloadSettings#76f36233 flags:# low:flags.0?true high:flags.1?true settings:AutoDownloadSetti
account.uploadTheme#1c3db333 flags:# file:InputFile thumb:flags.0?InputFile file_name:string mime_type:string =
account.createTheme#2b7ffd7f slug:string title:string document:InputDocument = Theme;
account.updateTheme#3b8ea202 flags:# format:string theme:InputTheme slug:flags.0?string title:flags.1?string do
account.saveTheme#f257106c theme:InputTheme unsave:Bool = Bool;
account.installTheme#7ae43737 flags:# dark:flags.0?true format:flags.1?string theme:flags.1?InputTheme = Bool;
account.getTheme#8d9d742b format:string theme:InputTheme document_id:long = Theme;
account.getThemes#285946f8 format:string hash:int = account.Themes;

users.getUsers#d91a548 id:Vector<InputUser> = Vector<User>;
users.getFullUser#ca30a5b1 id:InputUser = UserFull;
users.setSecureValueErrors#90c894b5 id:InputUser errors:Vector<SecureValueError> = Bool;

contacts.getContactIDs#2caa4a42 hash:int = Vector<int>;
contacts.getStatuses#c4a353ee = Vector<ContactStatus>;
contacts.getContacts#c023849f hash:int = contacts.Contacts;
contacts.importContacts#2c800be5 contacts:Vector<InputContact> = contacts.ImportedContacts;
contacts.deleteContacts#96a0e00 id:Vector<InputUser> = Updates;
contacts.deleteByPhones#1013fd9e phones:Vector<string> = Bool;
contacts.block#332b49fc id:InputUser = Bool;
contacts.unblock#e54100bd id:InputUser = Bool;
contacts.getBlocked#f57c350f offset:int limit:int = contacts.Blocked;
contacts.search#11f812d8 q:string limit:int = contacts.Found;
contacts.resolveUsername#f93ccba3 username:string = contacts.ResolvedPeer;
contacts.getTopPeers#d4982db5 flags:# correspondents:flags.0?true bots_pm:flags.1?true bots_inline:flags.2?true
contacts.resetTopPeerRating#1ae373ac category:TopPeerCategory peer:InputPeer = Bool;
contacts.resetSaved#879537f1 = Bool;
contacts.getSaved#82f1e39f = Vector<SavedContact>;
contacts.toggleTopPeers#8514bdda enabled:Bool = Bool;
contacts.addContact#e8f463d0 flags:# add_phone_privacy_exception:flags.0?true id:InputUser first_name:string la
contacts.acceptContact#f831a20f id:InputUser = Updates;
contacts.getLocated#a356056 geo_point:InputGeoPoint = Updates;
```

```
messages.getMessages#63c66506 id:Vector<InputMessage> = messages.Messages;
messages.getDialogs#a0ee3b73 flags:# exclude_pinned:flags.0?true folder_id:flags.1?int offset_date:int offset_i
messages.getHistory#dcbb8260 peer:InputPeer offset_id:int offset_date:int add_offset:int limit:int max_id:int m
messages.search#8614ef68 flags:# peer:InputPeer q:string from_id:flags.0?InputUser filter:MessagesFilter min_da
messages.readHistory#e306d3a peer:InputPeer max_id:int = messages.AffectedMessages;
messages.deleteHistory#1c015b09 flags:# just_clear:flags.0?true revoke:flags.1?true peer:InputPeer max_id:int =
messages.deleteMessages#e58e95d2 flags:# revoke:flags.0?true id:Vector<int> = messages.AffectedMessages;
messages.receivedMessages#5a954c0 max_id:int = Vector<ReceivedNotifyMessage>;
messages.setTyping#a3825e50 peer:InputPeer action:SendMessageAction = Bool;
messages.sendMessage#520c3870 flags:# no_webpage:flags.1?true silent:flags.5?true background:flags.6?true clear
messages.sendMedia#3491eba9 flags:# silent:flags.5?true background:flags.6?true clear_draft:flags.7?true peer:I
messages.forwardMessages#d9fee60e flags:# silent:flags.5?true background:flags.6?true with_my_score:flags.8?tru
messages.reportSpam#cf1592db peer:InputPeer = Bool;
messages.getPeerSettings#3672e09c peer:InputPeer = PeerSettings;
messages.report#bd82b658 peer:InputPeer id:Vector<int> reason:ReportReason = Bool;
messages.getChats#3c6aa187 id:Vector<int> = messages.Chats;
messages.getFullChat#3b831c66 chat_id:int = messages.ChatFull;
messages.editChatTitle#dc452855 chat_id:int title:string = Updates;
messages.editChatPhoto#ca4c79d8 chat_id:int photo:InputChatPhoto = Updates;
messages.addChatUser#f9a0aa09 chat_id:int user_id:InputUser fwd_limit:int = Updates;
messages.deleteChatUser#e0611f16 chat_id:int user_id:InputUser = Updates;
messages.createChat#9cb126e users:Vector<InputUser> title:string = Updates;
messages.getDhConfig#26cf8950 version:int random_length:int = messages.DhConfig;
messages.requestEncryption#f64daf43 user_id:InputUser random_id:int g_a:bytes = EncryptedChat;
messages.acceptEncryption#3dbc0415 peer:InputEncryptedChat g_b:bytes key_fingerprint:long = EncryptedChat;
messages.discardEncryption#edd923c5 chat_id:int = Bool;
messages.setEncryptedTyping#791451ed peer:InputEncryptedChat typing:Bool = Bool;
messages.readEncryptedHistory#7f4b690a peer:InputEncryptedChat max_date:int = Bool;
messages.sendEncrypted#a9776773 peer:InputEncryptedChat random_id:long data:bytes = messages.SentEncryptedMessa
messages.sendEncryptedFile#9a901b66 peer:InputEncryptedChat random_id:long data:bytes file:InputEncryptedFile =
messages.sendEncryptedService#32d439a4 peer:InputEncryptedChat random_id:long data:bytes = messages.SentEncrypt
messages.receivedQueue#55a5bb66 max_qts:int = Vector<long>;
messages.reportEncryptedSpam#4b0c8c0f peer:InputEncryptedChat = Bool;
messages.readMessageContents#36a73f77 id:Vector<int> = messages.AffectedMessages;
messages.getStickers#43d4f2c emoticon:string hash:int = messages.Stickers;
messages.getAllStickers#1c9618b1 hash:int = messages.AllStickers;
messages.getWebPagePreview#8b68b0cc flags:# message:string entities:flags.3?Vector<MessageEntity> = MessageMedi
messages.exportChatInvite#df7534c peer:InputPeer = ExportedChatInvite;
messages.checkChatInvite#3eadb1bb hash:string = ChatInvite;
messages.importChatInvite#6c50051c hash:string = Updates;
messages.getStickerSet#2619a90e stickerset:InputStickerSet = messages.StickerSet;
messages.installStickerSet#c78fe460 stickerset:InputStickerSet archived:Bool = messages.StickerSetInstallResult
messages.uninstallStickerSet#f96e55de stickerset:InputStickerSet = Bool;
messages.startBot#e6df7378 bot:InputUser peer:InputPeer random_id:long start_param:string = Updates;
messages.getMessagesViews#c4c8a55d peer:InputPeer id:Vector<int> increment:Bool = Vector<int>;
messages.editChatAdmin#a9e69f2e chat_id:int user_id:InputUser is_admin:Bool = Bool;
messages.migrateChat#15a3b8e3 chat_id:int = Updates;
messages.searchGlobal#bf7225a4 flags:# folder_id:flags.0?int q:string offset_rate:int offset_peer:InputPeer off
messages.reorderStickerSets#78337739 flags:# masks:flags.0?true order:Vector<long> = Bool;
messages.getDocumentByHash#338e2464 sha256:bytes size:int mime_type:string = Document;
messages.searchGifs#bf9a776b q:string offset:int = messages.FoundGifs;
messages.getSavedGifs#83bf3d52 hash:int = messages.SavedGifs;
messages.saveGif#327a30cb id:InputDocument unsave:Bool = Bool;
messages.getInlineBotResults#514e999d flags:# bot:InputUser peer:InputPeer geo_point:flags.0?InputGeoPoint quer
messages.setInlineBotResults#eb5ea206 flags:# gallery:flags.0?true private:flags.1?true query_id:long results:V
messages.sendInlineBotResult#220815b0 flags:# silent:flags.5?true background:flags.6?true clear_draft:flags.7?t
messages.getMessageEditData#fda68d36 peer:InputPeer id:int = messages.MessageEditData;
messages.editMessage#48f71778 flags:# no_webpage:flags.1?true peer:InputPeer id:int message:flags.11?string med
messages.editInlineBotMessage#83557dba flags:# no_webpage:flags.1?true id:InputBotInlineMessageID message:flags
messages.getBotCallbackAnswer#810a9fec flags:# game:flags.1?true peer:InputPeer msg_id:int data:flags.0?bytes =
messages.setBotCallbackAnswer#d58f130a flags:# alert:flags.1?true query_id:long message:flags.0?string url:flag
messages.getPeerDialogs#e470bcfd peers:Vector<InputDialogPeer> = messages.PeerDialogs;
messages.saveDraft#bc39e14b flags:# no_webpage:flags.1?true reply_to_msg_id:flags.0?int peer:InputPeer message:
messages.getAllDrafts#6a3f8d65 = Updates;
messages.getFeaturedStickers#2dacca4f hash:int = messages.FeaturedStickers;
messages.readFeaturedStickers#5b118126 id:Vector<long> = Bool;
messages.getRecentStickers#5ea192c9 flags:# attached:flags.0?true hash:int = messages.RecentStickers;
messages.saveRecentSticker#392718f8 flags:# attached:flags.0?true id:InputDocument unsave:Bool = Bool;
messages.clearRecentStickers#8999602d flags:# attached:flags.0?true = Bool;
messages.getArchivedStickers#57f17692 flags:# masks:flags.0?true offset_id:long limit:int = messages.ArchivedSt
messages.getMaskStickers#65b8c79f hash:int = messages.AllStickers;
messages.getAttachedStickers#cc5b67cc media:InputStickeredMedia = Vector<StickerSetCovered>;
messages.setGameScore#8ef8ecc0 flags:# edit_message:flags.0?true force:flags.1?true peer:InputPeer id:int user_
messages.setInlineGameScore#15ad9f64 flags:# edit_message:flags.0?true force:flags.1?true id:InputBotInlineMess
messages.getGameHighScores#e822649d peer:InputPeer id:int user_id:InputUser = messages.HighScores;
messages.getInlineGameHighScores#f635e1b id:InputBotInlineMessageID user_id:InputUser = messages.HighScores;
messages.getCommonChats#d0a48c4 user_id:InputUser max_id:int limit:int = messages.Chats;
messages.getAllChats#eba80ff0 except_ids:Vector<int> = messages.Chats;
messages.getWebPage#32ca8f91 url:string hash:int = WebPage;
messages.toggleDialogPin#a731e257 flags:# pinned:flags.0?true peer:InputDialogPeer = Bool;
messages.reorderPinnedDialogs#3b1adf37 flags:# force:flags.0?true folder_id:int order:Vector<InputDialogPeer> =
messages.getPinnedDialogs#d6b94df2 folder_id:int = messages.PeerDialogs;
messages.setBotShippingResults#e5f672fa flags:# query_id:long error:flags.0?string shipping_options:flags.1?Vec
messages.setBotPrecheckoutResults#9c2dd95 flags:# success:flags.1?true query_id:long error:flags.0?string = Boo
messages.uploadMedia#519bc2b1 peer:InputPeer media:InputMedia = MessageMedia;
messages.sendScreenshotNotification#c97df020 peer:InputPeer reply_to_msg_id:int random_id:long = Updates;
messages.getFavedStickers#21ce0b0e hash:int = messages.FavedStickers;
messages.faveSticker#b9ffc55b id:InputDocument unfave:Bool = Bool;
messages.getUnreadMentions#46578472 peer:InputPeer offset_id:int add_offset:int limit:int max_id:int min_id:int
messages.readMentions#f0189d3 peer:InputPeer = messages.AffectedHistory;
messages.getRecentLocations#bbc45b09 peer:InputPeer limit:int hash:int = messages.Messages;
messages.sendMultiMedia#cc0110cb flags:# silent:flags.5?true background:flags.6?true clear_draft:flags.7?true p
messages.uploadEncryptedFile#5057c497 peer:InputEncryptedChat file:InputEncryptedFile = EncryptedFile;
messages.searchStickerSets#c2b7d08b flags:# exclude_featured:flags.0?true q:string hash:int = messages.FoundSti
messages.getSplitRanges#1cff7e08 = Vector<MessageRange>;
messages.markDialogUnread#c286d98f flags:# unread:flags.0?true peer:InputDialogPeer = Bool;
messages.getDialogUnreadMarks#22e24e22 = Vector<DialogPeer>;
messages.clearAllDrafts#7e58ee9c = Bool;
messages.updatePinnedMessage#d2aaf7ec flags:# silent:flags.0?true peer:InputPeer id:int = Updates;
messages.sendVote#10ea6184 peer:InputPeer msg_id:int options:Vector<bytes> = Updates;
messages.getPollResults#73bb643b peer:InputPeer msg_id:int = Updates;
messages.getOnlines#6e2be050 peer:InputPeer = ChatOnlines;
messages.getStatsURL#812c2ae6 flags:# dark:flags.0?true peer:InputPeer params:string = StatsURL;
messages.editChatAbout#def60797 peer:InputPeer about:string = Bool;
messages.editChatDefaultBannedRights#a5866b41 peer:InputPeer banned_rights:ChatBannedRights = Updates;
messages.getEmojiKeywords#35a0e062 lang_code:string = EmojiKeywordsDifference;
messages.getEmojiKeywordsDifference#1508b6af lang_code:string from_version:int = EmojiKeywordsDifference;
messages.getEmojiKeywordsLanguages#4e9963b2 lang_codes:Vector<string> = Vector<EmojiLanguage>;
messages.getEmojiURL#d5b10c26 lang_code:string = EmojiURL;
messages.getSearchCounters#732eef00 peer:InputPeer filters:Vector<MessagesFilter> = Vector<messages.SearchCount
messages.requestUrlAuth#e33f5613 peer:InputPeer msg_id:int button_id:int = UrlAuthResult;
messages.acceptUrlAuth#f729ea98 flags:# write_allowed:flags.0?true peer:InputPeer msg_id:int button_id:int = Ur
messages.hidePeerSettingsBar#4facb138 peer:InputPeer = Bool;
messages.getScheduledHistory#e2c2685b peer:InputPeer hash:int = messages.Messages;
messages.getScheduledMessages#bdbb0464 peer:InputPeer id:Vector<int> = messages.Messages;
messages.sendScheduledMessages#bd38850a peer:InputPeer id:Vector<int> = Updates;
messages.deleteScheduledMessages#59ae2b16 peer:InputPeer id:Vector<int> = Updates;

updates.getState#edd4882a = updates.State;
updates.getDifference#25939651 flags:# pts:int pts_total_limit:flags.0?int date:int qts:int = updates.Differenc
updates.getChannelDifference#3173d78 flags:# force:flags.0?true channel:InputChannel filter:ChannelMessagesFilt

photos.updateProfilePhoto#f0bb5152 id:InputPhoto = UserProfilePhoto;
photos.uploadProfilePhoto#4f32c098 file:InputFile = photos.Photo;
photos.deletePhotos#87cf7f2f id:Vector<InputPhoto> = Vector<long>;
photos.getUserPhotos#91cd32a8 user_id:InputUser offset:int max_id:long limit:int = photos.Photos;

upload.saveFilePart#b304a621 file_id:long file_part:int bytes:bytes = Bool;
upload.getFile#b15a9afc flags:# precise:flags.0?true location:InputFileLocation offset:int limit:int = upload.F
upload.saveBigFilePart#de7b673d file_id:long file_part:int file_total_parts:int bytes:bytes = Bool;
upload.getWebFile#24e6818d location:InputWebFileLocation offset:int limit:int = upload.WebFile;
upload.getCdnFile#2000bcc3 file_token:bytes offset:int limit:int = upload.CdnFile;
upload.reuploadCdnFile#9b2754a8 file_token:bytes request_token:bytes = Vector<FileHash>;
upload.getCdnFileHashes#4da54231 file_token:bytes offset:int = Vector<FileHash>;
upload.getFileHashes#c7025931 location:InputFileLocation offset:int = Vector<FileHash>;

help.getConfig#c4f9186b = Config;
help.getNearestDc#1fb33026 = NearestDc;
help.getAppUpdate#522d5a7d source:string = help.AppUpdate;
help.getInviteText#4d392343 = help.InviteText;
help.getSupport#9cdf08cd = help.Support;
help.getAppChangelog#9010ef6f prev_app_version:string = Updates;
help.setBotUpdatesStatus#ec22cfcd pending_updates_count:int message:string = Bool;
help.getCdnConfig#52029342 = CdnConfig;
help.getRecentMeUrls#3dc0f114 referer:string = help.RecentMeUrls;
help.getProxyData#3d7758e1 = help.ProxyData;
help.getTermsOfServiceUpdate#2ca51fd1 = help.TermsOfServiceUpdate;
help.acceptTermsOfService#ee72f79a id:DataJSON = Bool;
help.getDeepLinkInfo#3fedc75f path:string = help.DeepLinkInfo;
help.getAppConfig#98914110 = JSONValue;
help.saveAppLog#6f02f748 events:Vector<InputAppEvent> = Bool;
help.getPassportConfig#c661ad08 hash:int = help.PassportConfig;
help.getSupportName#d360e72c = help.SupportName;
help.getUserInfo#38a08d3 user_id:InputUser = help.UserInfo;
help.editUserInfo#66b91b70 user_id:InputUser message:string entities:Vector<MessageEntity> = help.UserInfo;

channels.readHistory#cc104937 channel:InputChannel max_id:int = Bool;
channels.deleteMessages#84c1fd4e channel:InputChannel id:Vector<int> = messages.AffectedMessages;
channels.deleteUserHistory#d10dd71b channel:InputChannel user_id:InputUser = messages.AffectedHistory;
channels.reportSpam#fe087810 channel:InputChannel user_id:InputUser id:Vector<int> = Bool;
channels.getMessages#ad8c9a23 channel:InputChannel id:Vector<InputMessage> = messages.Messages;
channels.getParticipants#123e05e9 channel:InputChannel filter:ChannelParticipantsFilter offset:int limit:int ha
channels.getParticipant#546dd7a6 channel:InputChannel user_id:InputUser = channels.ChannelParticipant;
channels.getChannels#a7f6bbb id:Vector<InputChannel> = messages.Chats;
channels.getFullChannel#8736a09 channel:InputChannel = messages.ChatFull;
channels.createChannel#3d5fb10f flags:# broadcast:flags.0?true megagroup:flags.1?true title:string about:string
channels.editAdmin#d33c8902 channel:InputChannel user_id:InputUser admin_rights:ChatAdminRights rank:string = U
channels.editTitle#566decd0 channel:InputChannel title:string = Updates;
channels.editPhoto#f12e57c9 channel:InputChannel photo:InputChatPhoto = Updates;
channels.checkUsername#10e6bd2c channel:InputChannel username:string = Bool;
channels.updateUsername#3514b3de channel:InputChannel username:string = Bool;
channels.joinChannel#24b524c5 channel:InputChannel = Updates;
channels.leaveChannel#f836aa95 channel:InputChannel = Updates;
channels.inviteToChannel#199f3a6c channel:InputChannel users:Vector<InputUser> = Updates;
channels.deleteChannel#c0111fe3 channel:InputChannel = Updates;
channels.exportMessageLink#ceb77163 channel:InputChannel id:int grouped:Bool = ExportedMessageLink;
channels.toggleSignatures#1f69b606 channel:InputChannel enabled:Bool = Updates;
channels.getAdminedPublicChannels#f8b036af flags:# by_location:flags.0?true check_limit:flags.1?true = messages
channels.editBanned#72796912 channel:InputChannel user_id:InputUser banned_rights:ChatBannedRights = Updates;
channels.getAdminLog#33ddf480 flags:# channel:InputChannel q:string events_filter:flags.0?ChannelAdminLogEvents
channels.setStickers#ea8ca4f9 channel:InputChannel stickerset:InputStickerSet = Bool;
channels.readMessageContents#eab5dc38 channel:InputChannel id:Vector<int> = Bool;
channels.deleteHistory#af369d42 channel:InputChannel max_id:int = Bool;
channels.togglePreHistoryHidden#eabbb94c channel:InputChannel enabled:Bool = Updates;
channels.getLeftChannels#8341ecc0 offset:int = messages.Chats;
channels.getGroupsForDiscussion#f5dad378 = messages.Chats;
channels.setDiscussionGroup#40582bb2 broadcast:InputChannel group:InputChannel = Bool;
channels.editCreator#8f38cd1f channel:InputChannel user_id:InputUser password:InputCheckPasswordSRP = Updates;
channels.editLocation#58e63f6d channel:InputChannel geo_point:InputGeoPoint address:string = Bool;
channels.toggleSlowMode#edd49ef0 channel:InputChannel seconds:int = Updates;

bots.sendCustomRequest#aa2769ed custom_method:string params:DataJSON = DataJSON;
bots.answerWebhookJSONQuery#e6213f4d query_id:long data:DataJSON = Bool;

payments.getPaymentForm#99f09745 msg_id:int = payments.PaymentForm;
payments.getPaymentReceipt#a092a980 msg_id:int = payments.PaymentReceipt;
payments.validateRequestedInfo#770a8e74 flags:# save:flags.0?true msg_id:int info:PaymentRequestedInfo = paymen
payments.sendPaymentForm#2b8879b3 flags:# msg_id:int requested_info_id:flags.0?string shipping_option_id:flags.
payments.getSavedInfo#227d824b = payments.SavedInfo;
payments.clearSavedInfo#d83d70c1 flags:# credentials:flags.0?true info:flags.1?true = Bool;

stickers.createStickerSet#9bd86e6a flags:# masks:flags.0?true user_id:InputUser title:string short_name:string
stickers.removeStickerFromSet#f7760f51 sticker:InputDocument = messages.StickerSet;
stickers.changeStickerPosition#ffb6d4ca sticker:InputDocument position:int = messages.StickerSet;
stickers.addStickerToSet#8653febe stickerset:InputStickerSet sticker:InputStickerSetItem = messages.StickerSet;

phone.getCallConfig#55451fa9 = DataJSON;
phone.requestCall#42ff96ed flags:# video:flags.0?true user_id:InputUser random_id:int g_a_hash:bytes protocol:P
phone.acceptCall#3bd2b4a0 peer:InputPhoneCall g_b:bytes protocol:PhoneCallProtocol = phone.PhoneCall;
phone.confirmCall#2efe1722 peer:InputPhoneCall g_a:bytes key_fingerprint:long protocol:PhoneCallProtocol = phon
phone.receivedCall#17d54f61 peer:InputPhoneCall = Bool;
phone.discardCall#b2cbc1c0 flags:# video:flags.0?true peer:InputPhoneCall duration:int reason:PhoneCallDiscardR
phone.setCallRating#59ead627 flags:# user_initiative:flags.0?true peer:InputPhoneCall rating:int comment:string
phone.saveCallDebug#277add7e peer:InputPhoneCall debug:DataJSON = Bool;

langpack.getLangPack#f2f2330a lang_pack:string lang_code:string = LangPackDifference;
langpack.getStrings#efea3803 lang_pack:string lang_code:string keys:Vector<string> = Vector<LangPackString>;
langpack.getDifference#cd984aa5 lang_pack:string lang_code:string from_version:int = LangPackDifference;
langpack.getLanguages#42c6978f lang_pack:string = Vector<LangPackLanguage>;
langpack.getLanguage#6a596502 lang_pack:string lang_code:string = LangPackLanguage;

folders.editPeerFolders#6847d0ab folder_peers:Vector<InputFolderPeer> = Updates;
folders.deleteFolder#1c295881 folder_id:int = Updates;
```

Telegram

Telegram is a cloud-based mobile and desktop messaging app with a focus on security and speed.

About

FAQ
Blog
Jobs

Mobile Apps

iPhone/iPad
Android
Windows Phone

Desktop Apps

PC/Mac/Linux
macOS
Web-browser

Platform

API
Translations
Instant View

Perfect Forward Secrecy

Related articles

Perfect Forward Secrecy in Secret Chats
Security guidelines for developers

> *This article is about Perfect Forward Secrecy in cloud chats, see also PFS in Secret Chats.*

Telegram supports Perfect Forward Secrecy (PFS).

To make this possible, the client generates a permanent authorization key using **p_q_inner_data** and a temporary key using **p_q_inner_data_temp**. (See Creating an Authorization Key for more info.) These 2 operations may be done in parallel and even using the same connection. The client must save an **expires_at** unix timestamp `expires_at = time + expires_in`.

Important: in order to achieve PFS, the client must **never** use the permanent auth_key_id directly. Every message that is sent to MTProto, must be encrypted by a **temp_auth_key_id**, that was bound to the **perm_auth_key_id**.

An unbound **temp_auth_key_id** may only be used with the following methods:

- auth.bindTempAuthKey
- help.getConfig
- help.getNearestDc

In order to bind a temporary authorization key to the permanent key the client creates a special binding message and executes the auth.bindTempAuthKey method using **temp_auth_key**. Once **auth.bindTempAuthKey** has been executed successfully, the client may signUp / signIn using other auth.* methods and continue using the API as usual; the client must also rewrite client info using initConnection after each binding. Each permanent key may only be bound to **one** temporary key at a time, binding a new temporary key overwrites the previous one.

Once the temporary key expires, the client needs to generate a new temporary key using **p_q_inner_data_temp**. Then it needs to re-bind that new temporary key to the initial permanent key. A new key can also be generated in advance, so that the client has a new key ready by the time the old one has expired.

For additional security, the client can store the temporary authorization key in RAM only and never save it in persistent storage.

A temporary authorization key may expire at any moment before **expires_at**, since such keys are also stored only in the RAM on the server-side. Be prepared to handle resulting MTProto errors correctly (non-existent auth_key_id results in a 404 error).

Client-Side Optimization

Simplified Acknowledgment of Message Delivery

An outgoing message may be considered sent once the server has assigned it an identifier. Normally, a client would learn of this from the result of the messages.sendMessage method.
The MTProto server provides a mechanism for "quick acknowledgments". Upon receiving such an acknowledgment, the client may be certain that the call to the send message method has at least been fully received by the server and placed in a processing queue, and can inform the user that the delivery was successful.
It is possible that the server's actual response will never be received by the client (an interrupted connection; or the app restarts at exactly the wrong time). To correctly handle these situations, you can use a special type of notification generated by the server when updates.getDifference is called: updateMessageID. When processing this notification, the client can use the **random_id** identifier to associate the previously transmitted message with the one delivered to the server.
If such a notification is not issued when updates.getDifference is called for one of the previously sent messages, the message must be marked as undelivered.

Server Salt

Server salt is a 64-bit number added to every outgoing and incoming message. At present, a single salt's lifespan is 1 hour, following which it is considered invalid and the server will return an error for all the messages that contain it. The error message will contain the correct salt, which may be immediately used for sending. Given this approach, there will always be a period of waiting before the client receives a new salt if it connects to the server less frequently than once an hour.
For improved performance, there is a special get_future_salts method, which fetches in advance a list of the salts that will be valid during the course of a specified period of time following the call (1 day, for example). A start time and an end time are specified for each salt. The salts overlap one another by half an hour. We recommend always using the record with the longest remaining lifespan.

Downloading Files and Uploading Data to the Server

We recommend that separate connections and sessions be created for these tasks. Remember that the extra sessions must be deleted when no longer needed.
It makes sense to download files over several connections (optimally to have a pool). When uploading data to a server one connection is enough to achieve the best results.

The file handling API is designed to perform data operations in parts. In its simplest implementation, the process of uploading files to a server looks like this: send a query, wait for a response, send the next query, etc. This approach does not optimize the use of network resources and the ping time has a huge effect.
The upload and download process is optimal when two or more queries are continuously being executed through one connection. In this arrangement, uploading to the server would look like this:

1. Send Query 1
2. Send Query 2
3. Wait for a response to Query 1
4. Send Query 3
5. Wait for a response to Query 2
6. Send Query 4
7. etc.

This will help reduce the effect of ping latency and maximize the channel workload.

Sending Messages in Bulk

Sometimes a client needs to transmit several send message method calls to the server all at once in a single message or in several consecutive messages. However, the server may execute these requests out of order (queries are handled by different servers to improve performance, which introduces a degree of randomness to the process).
This requires that dependencies be explicitly stated when processing queries by using the function

```
invokeAfterMsg#cb9f372d {X:Type} msg_id:long query:!X = X;
```

Actually, this means padding the beginning of the query with the 32-bit number `0xcb9f372d` and the 64-bit message identifier of the query on which the current query is dependent.

Grouping Updates

Generating updates (notifications about various server events) and delivering them to the client form two different parts of the system (respectively, the messenger API and MTProto). By itself, MTProto cannot modify in any way the data transmitted to the client, and the server API cannot respond to client-MTProto connection events.
Imagine the situation where a client loses its connection (or is intentionally disconnected from the network) for some time. If lots of different events occur before a new connection is established (contacts come online, typing event messages are sent), then when a connection is established the client will receive lots of data containing all of the intervening events, despite the fact that most of the data is obsolete.
The grouping of messages has been introduced to optimize such situations. If new events occur and the client has not managed to "collect" the previously generated updates, then the server API can combine them into a single package.

A client is able to control when the MTProto server begins to consider that the connection has been lost and grouping can begin (the earlier this occurs when there is no connection, the better for the client). This functionality is implemented through a special type of Ping message, ping_delay_disconnect, which specifies a time delay following which the server will close the current connection and start grouping messages.

It makes sense to combine the transmission of ping_delay_disconnect with that of other recurring tasks, such as updating the user status (account.updateStatus).

Setting the Typing Status

If a contact is not online, there is no need to invoke messages.setTyping.

Methods

Accepting the Terms of Service

Name	Description
help.getTermsOfServiceUpdate	Look for updates of telegram's terms of service
help.acceptTermsOfService	Accept the new terms of service

Dealing with spam and ToS violations

Name	Description
account.reportPeer	Report a peer for violation of telegram's Terms of Service
channels.reportSpam	Reports some messages from a user in a supergroup as spam; requires administrator rights in the supergroup
messages.report	Report a message in a chat for violation of telegram's Terms of Service
messages.reportSpam	Report a new incoming chat for spam, if the peer settings of the chat allow us to do that
messages.reportEncryptedSpam	Report a secret chat for spam

Fetching configuration

Name	Description
help.getAppChangelog	Get changelog of current app
help.getAppConfig	Get app-specific configuration
help.getAppUpdate	Returns information on update availability for the current application.
help.getConfig	Returns current configuration, icluding data center configuration.
help.getInviteText	Returns text of a text message with an invitation.
help.getNearestDc	Returns info on data centre nearest to the user.
help.getSupport	Returns the support user for the 'ask a question' feature.
help.getSupportName	Get localized name of the telegram support user

Miscellaneous

Name	Description
help.saveAppLog	Saves logs of application on the server.
initConnection	Initialize connection
invokeAfterMsg	Invokes a query after successfull completion of one of the previous queries.
invokeAfterMsgs	Invokes a query after a successfull completion of previous queries
invokeWithLayer	Invoke the specified query using the specified API layer
invokeWithoutUpdates	Invoke a request without subscribing the used connection for updates (this is enabled by default for file queries).

Registration/Authorization

Name	Description
auth.bindTempAuthKey	Binds a temporary authorization key `temp_auth_key_id` to the permanent authorization key `perm_auth_key_id`. Each permanent key may only be bound to one temporary key at a time, binding a new temporary key overwrites the previous one. For more information, see Perfect Forward Secrecy.
auth.cancelCode	Cancel the login verification code
auth.checkPassword	Try logging to an account protected by a 2FA password.
auth.dropTempAuthKeys	Delete all temporary authorization keys **except for** the ones specified
auth.exportAuthorization	Returns data for copying authorization to another data-centre.
auth.importAuthorization	Logs in a user using a key transmitted from his native data-centre.
auth.importBotAuthorization	Login as a bot
auth.logOut	Logs out the user.
auth.recoverPassword	Reset the 2FA password using the recovery code sent using auth.requestPasswordRecovery.
auth.requestPasswordRecovery	Request recovery code of a 2FA password, only for accounts with a recovery email configured.
auth.resendCode	Resend the login code via another medium, the phone code type is determined by the return value of the previous auth.sendCode/auth.resendCode: see login for more info.
auth.resetAuthorizations	Terminates all user's authorized sessions except for the current one. After calling this method it is necessary to reregister the current device using the method account.registerDevice
auth.sendCode	Send the verification code for login
auth.signIn	Signs in a user with a validated phone number.
auth.signUp	Registers a validated phone number in the system.

Working with GDPR export

Name	Description
account.initTakeoutSession	Intialize account takeout session
account.finishTakeoutSession	Finish account takeout session
messages.getSplitRanges	Get message ranges for saving the user's chat history
channels.getLeftChannels	Get a list of channels/supergroups we left
invokeWithMessagesRange	Invoke with the given message range
invokeWithTakeout	Invoke a method within a takeout session

Working with GIFs (actually MPEG4 GIFs)

Name	Description
messages.getSavedGifs	Get saved GIFs
messages.saveGif	Add GIF to saved gifs list
messages.searchGifs	Search for GIFs

Working with TSF (internal use only)

Name	Description
help.editUserInfo	Internal use
help.getUserInfo	Internal use

Working with 2FA login

Name	Description
account.confirmPasswordEmail	Verify an email to use as 2FA recovery method.
account.resendPasswordEmail	Resend the code to verify an email to use as 2FA recovery method.
account.cancelPasswordEmail	Cancel the code that was sent to verify an email to use as 2FA recovery method.
account.getPassword	Obtain configuration for two-factor authorization with password
account.getPasswordSettings	Get private info associated to the password info (recovery email, telegram passport info & so on)
account.updatePasswordSettings	Set a new 2FA password

Working with Seamless Telegram Login

Name	Description
messages.requestUrlAuth	Get more info about a Seamless Telegram Login authorization request, for more info click here »
messages.acceptUrlAuth	Use this to accept a Seamless Telegram Login authorization request, for more info click here »
account.getWebAuthorizations	Get web login widget authorizations
account.resetWebAuthorization	Log out an active web telegram login session
account.resetWebAuthorizations	Reset all active web telegram login sessions

Working with VoIP calls

Name	Description
phone.acceptCall	Accept incoming call
phone.confirmCall	Complete phone call E2E encryption key exchange »
phone.discardCall	Refuse or end running call
phone.getCallConfig	Get phone call configuration to be passed to libtgvoip's shared config
phone.receivedCall	Optional: notify the server that the user is currently busy in a call: this will automatically refuse all incoming phone calls until the current phone call is ended.
phone.requestCall	Start a telegram phone call
phone.saveCallDebug	Send phone call debug data to server
phone.setCallRating	Rate a call

Working with channels/supergroups/geogroups

Name	Description
channels.createChannel	Create a supergroup/channel.
channels.deleteChannel	Delete a channel/supergroup
channels.deleteHistory	Delete the history of a supergroup
channels.deleteMessages	Delete messages in a channel/supergroup
channels.deleteUserHistory	Delete all messages sent by a certain user in a supergroup
channels.editAdmin	Modify the admin rights of a user in a supergroup/channel.
channels.editBanned	Ban/unban/kick a user in a supergroup/channel.
channels.editCreator	Transfer channel ownership
channels.editLocation	Edit location of geogroup
channels.editPhoto	Change the photo of a channel/supergroup
channels.editTitle	Edit the name of a channel/supergroup
channels.exportMessageLink	Get link and embed info of a message in a channel/supergroup
channels.getAdminLog	Get the admin log of a channel/supergroup

channels.getAdminedPublicChannels	Get channels/supergroups/geogroups we're admin in. Usually called when the user exceeds the limit for owned public channels/supergroups/geogroups, and the user is given the choice to remove one of his channels/supergroups/geogroups.
channels.getChannels	Get info about channels/supergroups
channels.getFullChannel	Get full info about a channel
channels.getGroupsForDiscussion	Get all groups that can be used as discussion groups
channels.getMessages	Get channel/supergroup messages
channels.getParticipant	Get info about a channel/supergroup participant
channels.getParticipants	Get the participants of a channel
channels.inviteToChannel	Invite users to a channel/supergroup
channels.joinChannel	Join a channel/supergroup
channels.leaveChannel	Leave a channel/supergroup
channels.readHistory	Mark channel/supergroup history as read
channels.readMessageContents	Mark channel/supergroup message contents as read
channels.setDiscussionGroup	Associate a group to a channel as discussion group for that channel
channels.setStickers	Associate a stickerset to the supergroup
channels.togglePreHistoryHidden	Hide/unhide message history for new channel/supergroup users
channels.toggleSignatures	Enable/disable message signatures in channels
channels.toggleSlowMode	Toggle supergroup slow mode: if enabled, users will only be able to send one message every `seconds` seconds
messages.getStatsURL	Returns URL with the chat statistics. Currently this method can be used only for channels

Working with chats/supergroups/channels

Name	Description
messages.addChatUser	Adds a user to a chat and sends a service message on it.
messages.checkChatInvite	Check the validity of a chat invite link and get basic info about it
messages.createChat	Creates a new chat.
messages.deleteChatUser	Deletes a user from a chat and sends a service message on it.
messages.editChatAbout	Edit the description of a group/supergroup/channel.
messages.editChatAdmin	Make a user admin in a legacy group.
messages.editChatDefaultBannedRights	Edit the default banned rights of a channel/supergroup/group.
messages.editChatPhoto	Changes chat photo and sends a service message on it
messages.editChatTitle	Chanages chat name and sends a service message on it.
messages.exportChatInvite	Export an invite link for a chat
messages.getAllChats	Get all chats, channels and supergroups
messages.getChats	Returns chat basic info on their IDs.
messages.getCommonChats	Get chats in common with a user
messages.getFullChat	Returns full chat info according to its ID.
messages.importChatInvite	Import a chat invite and join a private chat/supergroup/channel
messages.migrateChat	Turn a legacy group into a supergroup

Working with deep links

Name	Description
messages.startBot	Start a conversation with a bot using a deep linking parameter
help.getDeepLinkInfo	Get info about a `t.me` link
help.getRecentMeUrls	Get recently used `t.me` links

Working with files

Name	Description
help.getCdnConfig	Get configuration for CDN file downloads.
upload.getCdnFile	Download a CDN file.
upload.getCdnFileHashes	Get SHA256 hashes for verifying downloaded CDN files
upload.reuploadCdnFile	Request a reupload of a certain file to a CDN DC.
upload.getFile	Returns content of a whole file or its part.
upload.getFileHashes	Get SHA256 hashes for verifying downloaded files
upload.getWebFile	Returns content of an HTTP file or a part, by proxying the request through telegram.
upload.saveBigFilePart	Saves a part of a large file (over 10Mb in size) to be later passed to one of the methods.
upload.saveFilePart	Saves a part of file for futher sending to one of the methods.
messages.uploadEncryptedFile	Upload encrypted file and associate it to a secret chat
messages.uploadMedia	Upload a file and associate it to a chat (without actually sending it to the chat)
messages.getDocumentByHash	Get a document by its SHA256 hash, mainly used for gifs

Working with instant view pages

Name	Description
messages.getWebPage	Get instant view page

messages.getWebPage	Get instant view page
messages.getWebPagePreview	Get preview of webpage

Working with secret chats

Name	Description
messages.setEncryptedTyping	Send typing event by the current user to a secret chat.
messages.readEncryptedHistory	Marks message history within a secret chat as read.
messages.acceptEncryption	Confirms creation of a secret chat
messages.discardEncryption	Cancels a request for creation and/or delete info on secret chat.
messages.requestEncryption	Sends a request to start a secret chat to the user.
messages.sendEncrypted	Sends a text message to a secret chat.
messages.sendEncryptedFile	Sends a message with a file attachment to a secret chat
messages.sendEncryptedService	Sends a service message to a secret chat.
messages.getDhConfig	Returns configuration parameters for Diffie-Hellman key generation. Can also return a random sequence of bytes of required length.
messages.receivedQueue	Confirms receipt of messages in a secret chat by client, cancels push notifications.

Working with telegram passport

Name	Description
account.sendVerifyEmailCode	Send the verification email code for telegram passport.
account.verifyEmail	Verify an email address for telegram passport.
account.sendVerifyPhoneCode	Send the verification phone code for telegram passport.
account.verifyPhone	Verify a phone number for telegram passport.
account.acceptAuthorization	Sends a Telegram Passport authorization form, effectively sharing data with the service
account.getAuthorizationForm	Returns a Telegram Passport authorization form for sharing data with a service
account.getAuthorizations	Get logged-in sessions
account.deleteSecureValue	Delete stored Telegram Passport documents, for more info see the passport docs »
account.getAllSecureValues	Get all saved Telegram Passport documents, for more info see the passport docs »
account.getSecureValue	Get saved Telegram Passport document, for more info see the passport docs »
account.saveSecureValue	Securely save Telegram Passport document, for more info see the passport docs »
help.getPassportConfig	Get passport configuration
users.setSecureValueErrors	Notify the user that the sent passport data contains some errors The user will not be able to re-submit their Passport data to you until the errors are fixed (the contents of the field for which you returned the error must change). Use this if the data submitted by the user doesn't satisfy the standards your service requires for any reason. For example, if a birthday date seems invalid, a submitted document is blurry, a scan shows evidence of tampering, etc. Supply some details in the error message to make sure the user knows how to correct the issues.

Working with updates

Name	Description
updates.getChannelDifference	Returns the difference between the current state of updates of a certain channel and transmitted.
updates.getDifference	Get new updates.
updates.getState	Returns a current state of updates.

Working with bot inline queries and callback buttons

Name	Description
messages.getInlineBotResults	Query an inline bot
messages.setInlineBotResults	Answer an inline query, for bots only
messages.sendInlineBotResult	Send a result obtained using messages.getInlineBotResults.
messages.getBotCallbackAnswer	Press an inline callback button and get a callback answer from the bot
messages.setBotCallbackAnswer	Set the callback answer to a user button press (bots only)
messages.editInlineBotMessage	Edit an inline bot message

Working with bots (internal bot API use)

Name	Description
bots.answerWebhookJSONQuery	Answers a custom query; for bots only
bots.sendCustomRequest	Sends a custom request; for bots only
help.setBotUpdatesStatus	Informs the server about the number of pending bot updates if they haven't been processed for a long time; for bots only

Working with cloud themes

Name	Description
account.updateTheme	Update theme
account.uploadTheme	Upload theme
account.getThemes	Get installed themes